Scalable Time Series Similarity Search for Data Analytics

DISSERTATION

zur Erlangung des akademischen Grades
doctor rerum naturalium
(Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin

von
Dipl.-Inform. Patrick Schäfer

Präsidentin/Präsident der Humboldt-Universität zu Berlin
Prof. Dr. Jan-Hendrik Olbertz

Dekanin/Dekan der Mathematisch-Naturwissenschaftlichen Fakultät
Prof. Dr. Elmar Kulke

Gutachter/innen:

1. Prof. Dr. Alexander Reinefeld
2. Prof. Dr. Ulf Leser
3. Prof. Dr. Artur Andrzejak

Tag der mündlichen Prüfung: 19. Oktober 2015

Abstract

A time series is a collection of values sequentially recorded from sensors or live observations over time. Sensors for recording time series have become cheap and omnipresent. While data volumes explode, research in the field of time series data analytics has focused on the availability of (a) pre-processed and (b) moderately sized time series datasets in the last decades.

The analysis of real world datasets raises two major problems: Firstly, state-of-the-art similarity models require the time series to be pre-processed. Pre-processing aims at extracting approximately aligned characteristic subsequences and reducing noise. It is typically performed by a domain expert, may be more time consuming than the data mining part itself, and simply does not scale to large data volumes. Secondly, time series research has been driven by accuracy metrics and not by reasonable execution times for large data volumes. This results in quadratic to biquadratic computational complexities of state-of-the-art similarity models.

This dissertation addresses both issues by introducing a symbolic time series representation and three different similarity models. These contribute to state of the art by being pre-processing-free, noise-robust, and scalable.

Our experimental evaluation on 91 real-world and benchmark datasets shows that our methods provide higher accuracy for most datasets when compared to 15 state-of-the-art similarity models. Meanwhile they are up to three orders of magnitude faster, require less pre-processing for noise or alignment, or scale to large data volumes.

Zusammenfassung

Eine Zeitreihe ist eine zeitlich geordnete Folge von Datenpunkten. Zeitreihen werden typischerweise über Sensormessungen oder Experimente erfasst. Sensoren sind so preiswert geworden, dass sie praktisch allgegenwärtig sind. Während dadurch die Menge an Zeitreihen regelrecht explodiert, lag der Schwerpunkt der Forschung in den letzten Jahrzehnten auf der Analyse von (a) vorgefilterten und (b) kleinen Zeitreihendatensätzen.

Die Analyse realer Zeitreihendatensätze wirft zwei Probleme auf: Erstens setzen aktuelle Ähnlichkeitsmodelle eine Vorfilterung der Zeitreihen voraus. Das beinhaltet die Extraktion charakteristischer Teilsequenzen und das Entfernen von Rauschen. Diese Vorverarbeitung muss durch einen Spezialisten erfolgen. Sie kann zeit- und kostenintensiver als die anschließende Analyse und für große Datensätze unrentabel werden. Zweitens führte die Verbesserung der Genauigkeit aktueller Ähnlichkeitsmodelle zu einem unverhältnismäßig hohen Anstieg der Komplexität (quadratisch bis biquadratisch).

Diese Dissertation behandelt beide Probleme. Es wird eine symbolische Zeitreihenrepräsentation vorgestellt. Darauf aufbauend werden drei verschiedene Ähnlichkeitsmodelle eingeführt. Diese erweitern den aktuellen Stand der Forschung insbesondere dadurch, dass sie vorverarbeitungsfrei, unempfindlich gegenüber Rauschen und skalierbar sind.

Anhand von 91 realen Datensätzen und Benchmarkdatensätzen wird zusätzlich gezeigt, dass die hier eingeführten Modelle auf den meisten Datenätzen die höchste Genauigkeit im Vergleich zu 15 aktuellen Ähnlichkeitsmodellen liefern. Sie sind teilweise drei Größenordnungen schneller und benötigen kaum Vorfilterung.

Contents

Acronyms

APCA	Adaptive Piecewise Constant Approximation
BOP	Bag-Of-Patterns
BOSS	Bag-Of-SFA-Symbols
BOSS VS	Bag-Of-SFA-Symbols in Vector Space
CHEBY	Chebyshev Polynomials
DFT	Discrete Fourier Transform
DTW	Dynamic Time Warping
DTW CV	DTW with the warping window constraint set through Cross Validation
DWT	Discrete Wavelet Transform
ED	Euclidean Distance
FT	Fourier Transform
idf	inverse document frequency
iSAX	indexable SAX
MCB	Multiple Coefficient Binning
MFT	Momentary Fourier Transform
1-NN	1-Nearest-Neighbor
PAA	Piecewise Aggregate Approximation
PLA	Piecewise Linear Approximation
LCSS	Longest Common Subsequence
LSH	Locality Sensitive Hashing
SAM	Spatial Access Method
SAX	Symbolic Aggregate approXimation
SAX-VSM	Symbolic Aggregate approXimation Vector Space Model
SVD	Singular Value Decomposition
SVM	Support Vector Machine
tf-idf	term frequency-inverse document frequency
tf	term frequency
TS-tree	Time-Series-tree

Chapter 1

Introduction: It is about time.

> *"[...] research has not be driven so much by actual problems but by an interest in proposing new approaches.", Antunes and Oliveira, 2001 [9].*

Time series are a collection of values sequentially recorded from sensors or live observations over time. Time series are collected in application domains ranging from business forecasting (sales, stock market), medicine (ECG signals), biology (wildlife monitoring, weather), security (passgraph, intrusion detection), to scientific databases to name but a few examples. In the last decades there has been an enormous increase in data volumes. It is expected to reach 100 zettabytes (10^{21}) in 2020 [85]. At the same time, sensors for recording time series have become cheap and omnipresent as in RFID chips, wearable sensors (wrist bands, smartphones), smart homes (smart plugs [39], smart meters, fire alarms, temperature, security), or event-based systems (soccer player shoes [53]). A smart-meter with a sampling rate of one value per minute records more than 0.5 million values a year. Given a company has millions of customers, this easily accounts for trillions (10^{12}) of measurements per year. The goal is to extract knowledge from that raw data.

Data analytics based on similarity is the tool for exploring time series databases. The similarity of two time series is commonly defined by a similarity/distance measure: two time series are similar if their distance is small. While a human has an intuitive understanding of the similarity of two time series, this task may become very hard for a computer, i.e., in the case of bird songs in the presence of ambient noise. Working with time series is difficult as the definition of similarity depends on the application domain, the data analytics task, and the data may be erroneous, extraneous and constitute large databases. A human compares two time series by learning an abstract shape from the time series and comparing the similarity of the abstract patterns contained in the time series. At the core of each data analysis technique there are (a) a *time series representati*on and (b) a *similarity measure* to compare two time series. The difficulties arise from defining an abstract representation of time series that emphasizes the characteristic shape features of a time series, and designing a similarity measure that imitates the human perception of

the similarity of objects. This similarity measure should give a high similarity for similar time series even though they may not be identical mathematically [30].

Common distance measures are Dynamic Time Warping (DTW) or the Euclidean Distance (ED). There is a consensus that DTW is among the best similarity measures and should be used as the benchmark to compare to [13, 29, 49]. While both have proven their value, they are decades old and we believe these do not meet today's requirements. It has been established as a good research practice to compare every new method using the UCR time series classification benchmark datasets [42]. The over-dependance of research on these datasets has led to at least two potential pitfalls:

1. **Assuming pre-processed datasets**: Analyzing time series data from sensors is a challenging task, as the time series may be recorded at variable lengths, be erroneous, extraneous, or highly redundant due to repetitive (sub-)structures, and these may constitute large databases. In time series literature the UCR time series datasets are commonly used to underline the utility of new algorithms. These datasets were pre-processed by domain experts for *equivalent-length, approximately aligned substructures, and noise was filtered*, later referred to as *alignment*. It has been criticized that this pre-processing can be more time consuming than the data mining part itself, which led to research on time series analysis with reasonable assumptions [38]: there is a need for data mining methods that are *alignment-free* and can deal with erroneous or extraneous data. Searching for similarities in substructures (subsequence to subsequence alignment) rather than matching the time series as a whole is a promising starting point to deal with the alignment part. A time series representation based on signal processing techniques like digital filters can be used to deal with noise. The key challenge is to provide a similarity measure and a time series representation that are alignment-free and deal with erroneous and extraneous data by incorporating invariance (robustness) to noise. These should not be restricted to a certain application domain.

2. **Focus on accuracy rather than scalability**: Two trends have evolved in time series data analytics: the emergence of large datasets and machine learning on streaming data (aka real-time analytics). The first requires an algorithm able to handle large amounts of data in reasonable time. The latter is troublesome as the underlying source generating the data might change over time (non-stationary time series). This requires evolving the model of the data by incremental updates or frequently rebuilding the model. The UCR time series datasets have led to a wealth of time series classification algorithms. However, the over-dependence on these datasets is troublesome as the largest of those datasets (StarlightCurves) contains only several thousand time series. When it comes to classification for example, research has focused on accuracy[1] and overlooks the scalability in execution times. The classification time of the 1-NN DTW classifier is in the order of hours

[1] "Accuracy is, in our option, the most important [...].", Lines and Bagnall, 2014 [49]

on a single core to classify the StarlightCurves dataset using the state of the art implementation [62]. The 1-NN DTW is not even the slowest classifier proposed in literature. State of the art classifiers have a train complexity of up to $O(N^2n^3)$ and a test complexity of up to $O(Nn^2)$. The key challenge is to provide scalability in train and test times while maintaining high accuracy.

This dissertation addresses both before mentioned issues by introducing a symbolic time series representation called Symbolic Fourier Approximation (SFA) and three different time series similarity measures for scalable, alignment-free time series analytics in the presence of noise: (1) the Shotgun distance model, (2) the Bag-Of-SFA-Symbols (BOSS) model and (3) the Bag-Of-SFA-Symbols in Vector space (BOSS VS) model.

1.1 Contributions

1.1.1 A Symbolic Representation of Time Series

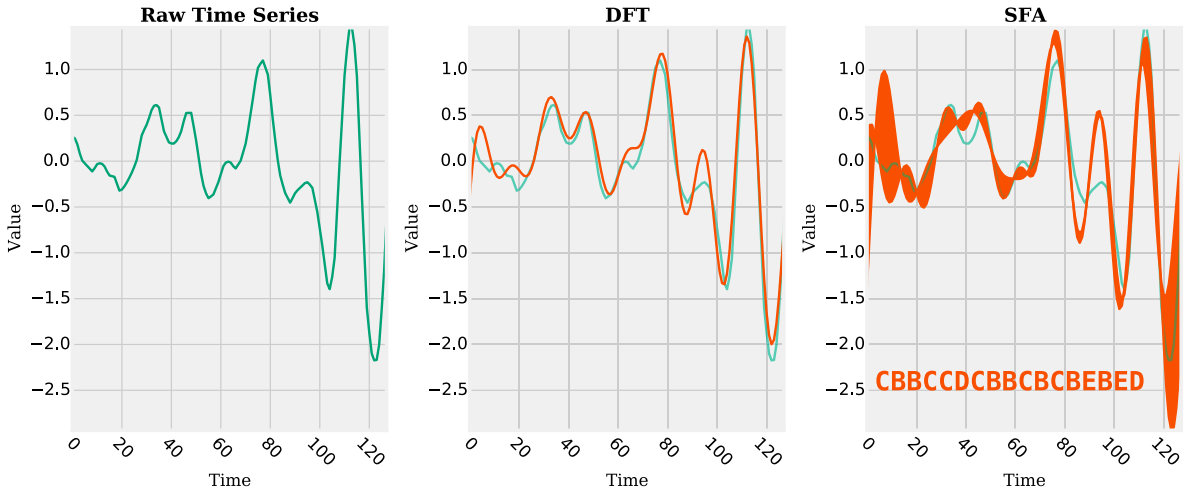


Figure 1.1 – A raw time series (left), the Fourier transform (center), and the SFA representation (right). The SFA transform results in a sequence of symbols (a string). This is in concept similar to a tight envelope around the Fourier transform of the time series.

A time series consisting of n measured values can be seen as a point in n-dimensional space, where the i-th measured value represents the i-th dimension. Dimensionality reduction aims at representing a time series by a lower dimensional representation, while retaining the significant features in the reduced representation. These can be divided into two groups: symbolic and numeric. In these methods a time series is represented as a sequence of discrete values (symbols) or real-world values respectively. Numeric representations apply noise reduction through signal processing techniques like approximation/ filtering/ compression. Symbolic representations add a second level of complexity

reduction that has an additional noise reducing effect by introducing a quantization step. Our symbolic representation of time series Symbolic Fourier Approximation (SFA) [76] represents each real valued time series by a sequence of symbols, named *SFA word*, using a finite alphabet of symbols. SFA is composed of *approximation* using the Fourier transform and *quantization* using a technique called Multiple Coefficient Binning (MCB) (Figure 1.1). The SFA transformation aims at:

- **Noise removal:** Rapidly changing sections of a signal are often associated with noise. These can be removed by using a low-pass filter. The SFA word length determines the number of Fourier coefficients and thereby the bandwidth of the low-pass filter.
- **String representation:** Using a string representation allows for string matching algorithms like hashing or the bag of words to be applied. The size of the alphabet determines the degree of quantization and has an additional noise reducing effect.
- **Frequency domain:** The SFA word length can be incrementally adapted by choosing an arbitrary subset of Fourier coefficients without recalculating the Fourier transform of the time series. Adding Fourier coefficients to an SFA word adds details and reduces the reconstruction error between the transformed and the original time series. This can be exploited in multiple ways such as variable length index construction or fast model updates for classification.
- **Dimensionality reduction:** Similarity search in data with increasing dimensionality results in an exponential growth of the search space, referred to as Curse of Dimensionality. A common approach to postpone this effect is to apply approximation to reduce the dimensionality of the original data prior to indexing. Our index structure SFA trie exploits the frequency domain nature of SFA as it grows in depth using variable prefix lengths of the SFA words for indexing.
- **Storage reduction:** SFA has a lower memory footprint than the original time series up to a factor of $1000 : 1$. By the use of (a) approximation the length of the time series is reduced by a factor of $100 : 1$, and (b) quantization only a few bits are needed to encode every symbol as opposed to a 8 byte double for real-values representations, leading to another reduction in size of up to $8 : 1$ to $32 : 1$.

The frequency domain nature makes SFA unique among the symbolic time series representations. Dynamically adding or removing Fourier coefficients to adapt the degree of approximation without recalculating the Fourier transform is at the core of all presented algorithms in this thesis.

In Chapter 3 we show that SFA is not only more accurate than the most common symbolic representation Symbolic Aggregate approXimation (SAX) [82, 47]. It can compete with the best numeric dimensionality reduction techniques. SFA allows for indexing large and high dimensional datasets through the SFA trie. In our experiments the SFA

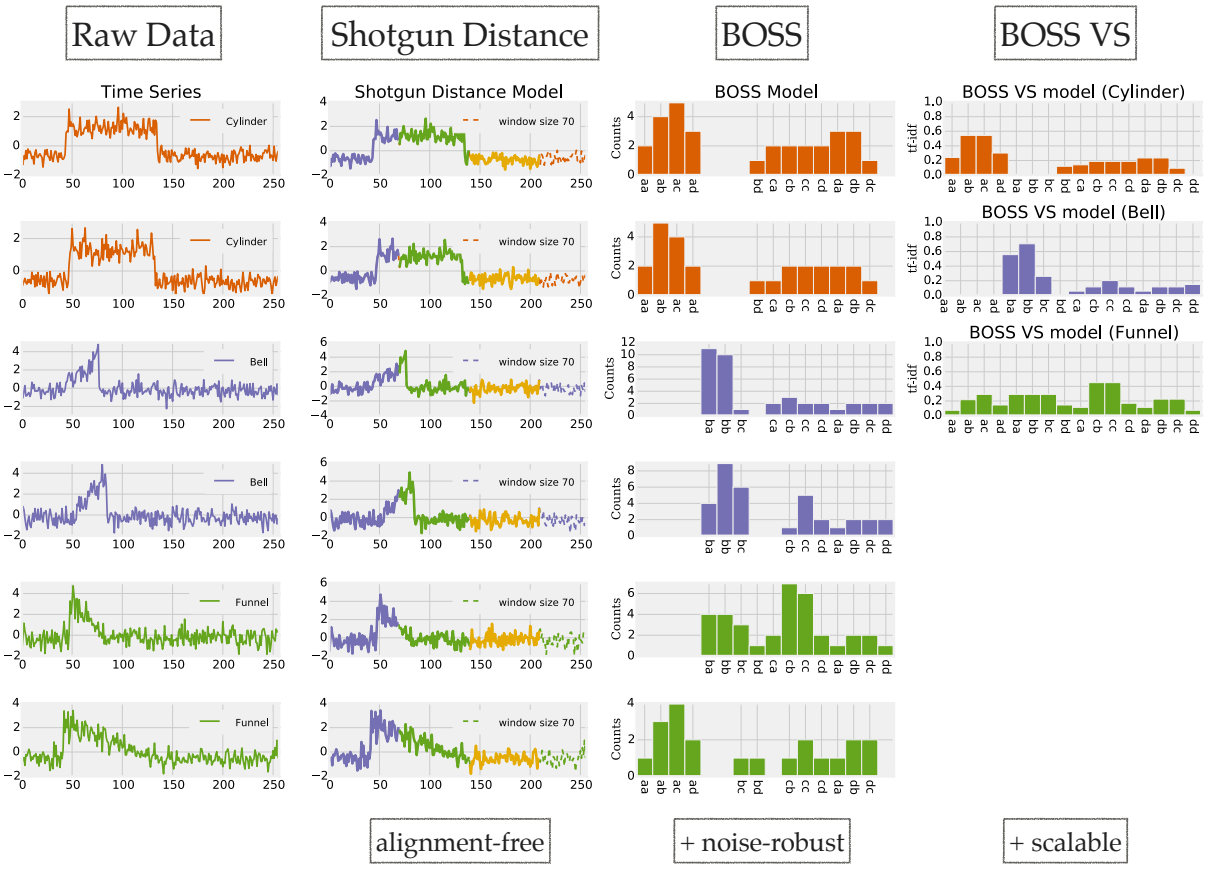


Figure 1.2 – Three time series models: Shotgun distance model, BOSS model, and BOSS VS model.

trie is the best spatial access method in terms of page accesses and wall-clock time on real and synthetic datasets.

1.1.2 Time Series Similarity Measures

The time series data analytics task is complicated by noise, dropouts, subtle distinctions, variable lengths, or extraneous data. A multitude of signals is composed of characteristic patterns. Consider human walking motions that are composed of gait cycles, or ECG signals that are composed of heart beats as concrete examples. Over the past decade algorithms were designed to mostly work with human assistance that is, to *extract these characteristic patterns and align and trim them for equivalent length and scaling* (the alignment). This might be a result of the over-dependence on the UCR datasets. We strive towards alignment-free, scalable time series analytics in the presence of noise by introducing three time series similarity measures:

Alignment-free: Common similarity measures require the data to be pre-processed by domain experts for equivalent-length, and approximately aligned substructures (*alignment*). Searching for similarities in substructures (subsequence to subsequence matching) rather than matching the time series as a whole is a promising starting point to deal with the alignment-free similarity of long time series. Our *Shotgun distance* [69, 71, 75] is based on subsequence to subsequence matching and thereby reduces the need for cost-ineffective pre-processing. The Shotgun distance breaks a query time series into subsequences and vertically aligns and horizontally scales/norms each subsequence prior to measuring the similarity to a sample time series. This significantly reduces the time and effort required for human pre-processing of a time series in order to extract and align characteristic segments.

Figure 1.2 shows the Shotgun distance model. A query time series is segmented into disjoint windows. For each query windows the similarity to the sample time series is separately measured by minimizing the Euclidean distance. We show the utility of our Shotgun distance on case studies in the context of bioacoustics, human motion detection, spectrographs or personalized medicine and compare it to state of the art in Chapter 4.

Alignment-free and noise-robust: Raw time series data may be recorded at variable lengths, or are composed of characteristic subsequences. These build a foundation for state of the art algorithms such as our Shotgun distance, or Shapelets [52, 91, 63]. However, extraneous or erroneous data, as a result of noise, have been paid surprisingly little attention to. Noise is commonly assumed to be filtered as part of a pre-processing step carried out by a human. Our *Bag-Of-SFA-Symbols* (BOSS) model [68] combines the noise tolerance of the time series representation Symbolic Fourier Approximation (SFA) with the structure-based representation of the bag-of-words model.

Figure 1.2 shows the BOSS model as a histogram over SFA words. It first extracts subsequences (patterns) from a time series. Next, it applies low-pass filtering and quantization to the subsequences using SFA which reduces noise and allows for string matching algorithms to be applied. Two time series are then compared based on the differences in the histogram of SFA words.

Apart from the invariance to noise, the BOSS model provides invariances (robustness) to phase shifts, offsets, amplitudes and occlusions by discarding the original ordering of the SFA words and normalization. This results in the highest classification and clustering accuracy in time series literature to date. We show that our BOSS ensemble classifier improves the best published classification accuracies in diverse application areas and on the public UCR classification benchmark datasets in Chapter 5.

Alignment-free, noise-robust, and scalable: When it comes to larger datasets, state of the art classifiers reach their limits because of unreasonable train or test times. State of the art algorithms have a quadratic to cubic computational complexity in the time series length. This results in execution times in the order of hours to classify moderately

sized datasets of a several thousand time series on a single core. In the context of mining large datasets and real-time data analytics there is a need for time series classification algorithms (a) with a low test time to allow for mining large datasets, (b) with tolerance to noise to provide high classification accuracy, (c) that are alignment-free, and (d) with a moderate train time to allow for frequent model updates. Our BOSS in Vector Space (BOSS VS) model [74] combines constant time classification and linear time training in the number of time series with high classification accuracy due to invariances to noise, phase shifts, offsets, amplitudes and occlusions.

Figure 1.2 illustrates the BOSS VS model. The BOSS VS model extends the BOSS model by a compact representation of classes instead of time series by using the *term frequency - inverse document frequency* (tf-idf) for each class. It significantly reduces the computational complexity and highlights characteristic SFA words by the use of the tf-idf weight matrix which provides an additional noise reducing effect.

The BOSS VS model beats competitors on use cases for noisy data and it is in the group of best state of the art classifiers with regard to classification accuracy on the UCR benchmark datasets in Chapter 6. The 1-NN BOSS VS is not the most accurate classifier, but it is (a) orders of magnitude faster than the 1-NN BOSS classifier, 1-NN DTW and state of the art, and (b) it is significantly more accurate than 1-NN DTW with or without a warping window. Its high speed combined with its good accuracy makes it unique and relevant for many practical use cases.

1.2 Outline of this Thesis

The thesis is organized as follows:

- Chapter 2 introduces the background on time series and data analytics.
- Chapter 3 presents the symbolic representation SFA and the SFA trie for indexing high dimensional datasets.
- Chapter 4 introduces our alignment-free Shotgun Distance for time series analysis without human pre-processing.
- In Chapter 5 we present our BOSS model for alignment-free time series analysis in the presence of noise.
- Chapter 6 introduces the BOSS VS model for alignment-free, noise-robust, and scalable time series classification.
- Chapter 7 draws a conclusion.

Chapter 2

Background

> *„A new algorithm is only of interest in terms of accuracy if it can significantly outperform 1-NN DTW with a full warping window". Bagnall and Lines, 2014 [13]*

Time series databases, resulting from recording data over time, range from meteorological data like sediments from drill holes [51], financial data like stock prices or product sales, biomedical and biochemical data like ECG signals [3], human walking motions [26], anthropology [90], security [52], astronomy, historical documents [90] insect wing beats [75], or cellular networks [59]. Unlike exact search, similarity based search finds results that are similar to a query based on some similarity measure. Examples of similarity queries include:

- Find all stocks that show *similar* trends,
- Find the *most unusual* heartbeat in a patient's ECG recording,
- Find *frequent patterns* in a bird sound recording,
- Find the patients with the most *similar* ECG recording,
- Find products with a *similar* sales pattern.

Figure 2.1 illustrates 12 different time series datasets. There is one example for each class in the datasets.

Why is time series similarity difficult?

Time series may be recorded at variable lengths, and are erroneous, extraneous due to noise, dropouts, subtle distinctions or highly redundant due to repetitive (sub-)structures. Figure 2.2 shows a dendrogram plot (binary tree), which illustrates the results of a hierarchical clustering. A dendrogram makes use of u-shapes that connect the two most

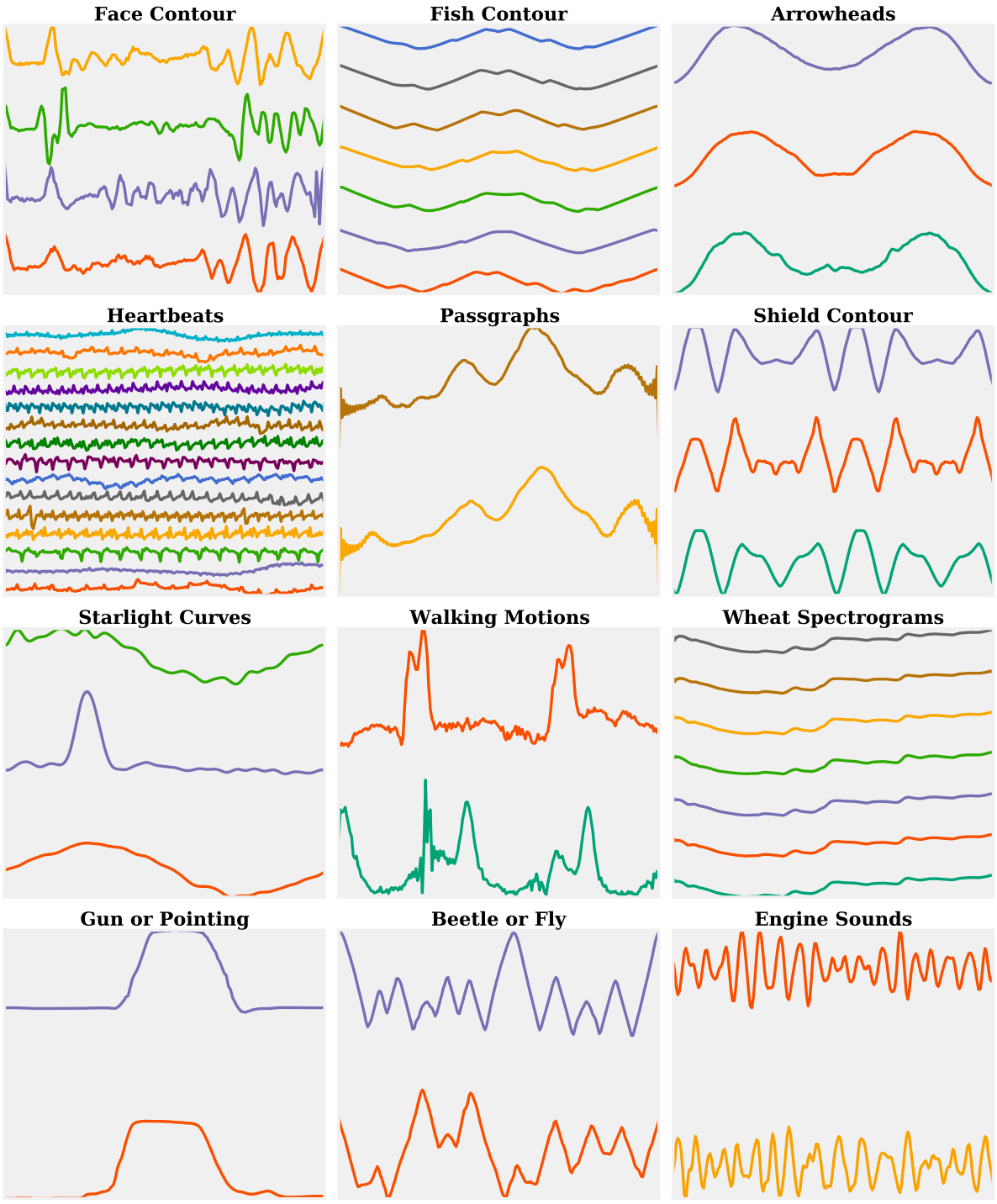


Figure 2.1 – Time Series Datasets: One sample for each class is shown.

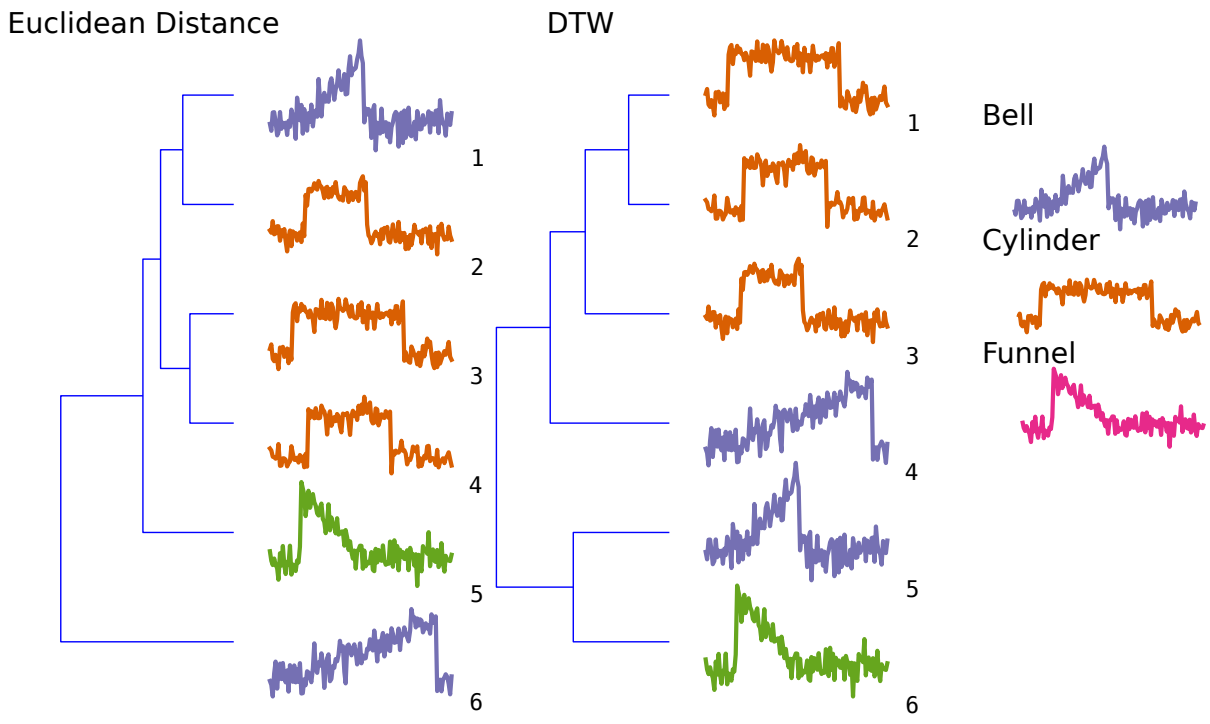


Figure 2.2 – Dendrogram of a hierarchical clustering of three different kinds of shapes: Bell, Cylinder, and Funnel.

similar time series. The height of each u-shape is equal to the distance (similarity). The figure shows a small example containing three simple shapes: Bells, Cylinders and Funnels. Each shape is distorted by Gaussian noise. This example illustrates some of the difficulties for time series similarity. For a human it is easy to extract the abstract shape to distinguish between each type of curve. The figure presents the dendrogram using two common similarity measures: Euclidean Distance (ED) and Dynamic Time Warping (DTW). For the human eye the results are very disappointing. Both, the ED and DTW fail to cluster the curves correctly, as similar shapes are in separate branches of the hierarchical clustering.

Working with time series is difficult as the definition of similarity depends on the application domain, the data analytics task, and the data may be distorted and constitute large databases. A human can easily abstract from distortions of a signal and extract a general model (the shape) of the time series. Based on this model, similarity can be determined. At the core of each data mining technique there are (a) a *time series representation (the data model)* and (b) a *similarity measure* to compare two time series. The difficulties arise from defining a representation that maintains the characteristic features of the time series, such as a human would extract the shape, and the definition of a similarity measure based on the human perception of similarity.

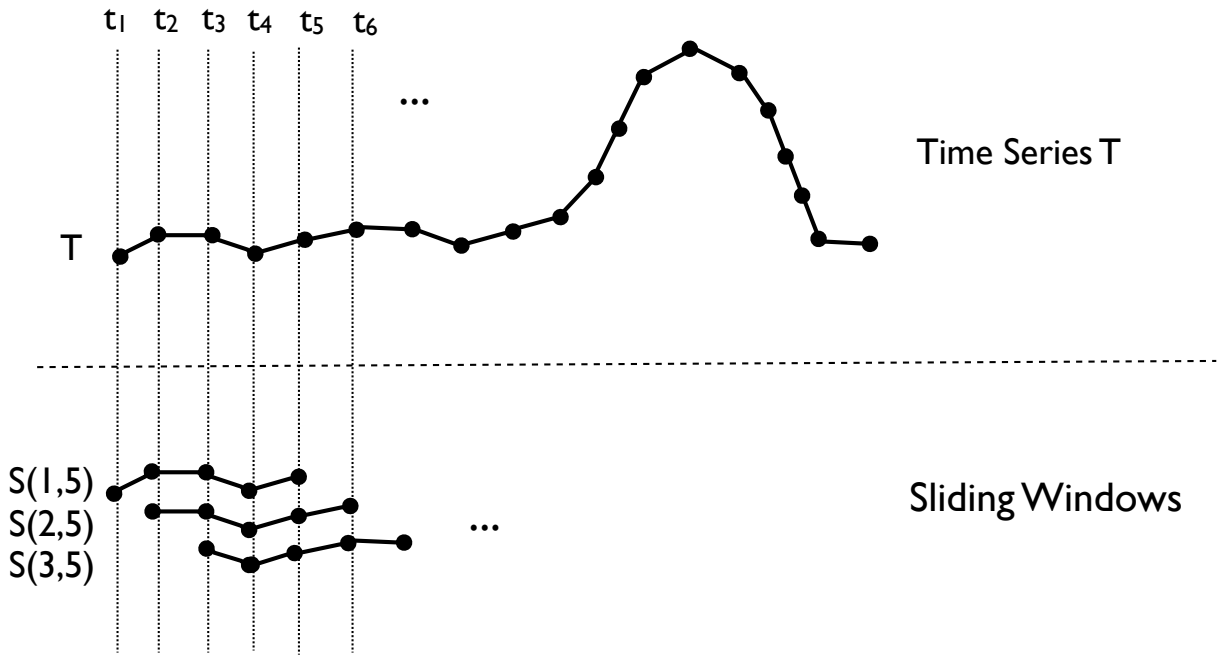


Figure 2.3 – Subsequences extracted from a time series using sliding windows.

2.1 Time Series

A *time series* consists of a sequence of n real values (omitting the time stamps):

$$T = (t_1, \ldots, t_n), \qquad t_i \epsilon \mathbb{R} \tag{2.1}$$

A *time series dataset* DS is an unordered set of N time series:

$$DS = \{T_1, \ldots, T_N\}, \qquad |DS| = N \tag{2.2}$$

A *time series subsequence* is a time series that omits some values from a longer time series but does not change the ordering of the remaining values. Given a time series T, a subsequence S is a time series with w contiguous values starting at offset a in T:

$$S(a, w) = (t_a, \ldots, t_{a+w-1}), \qquad with\ 1 \leq a \leq n - w + 1 \tag{2.3}$$

A subsequence at time interval i can be inferred from its predecessor $i - 1$ by one summation and subtraction, i.e., they share $w - 1$ values:

$$S(a, w) = S(a - 1, w) + x_a - x_{a-w}\ , \text{ for } a > 1 \tag{2.4}$$

Subsequences of fixed length w can be extracted from each offset in time series T (Figure 2.3). That is, a sliding window is shifted by an offset of *step*, *with* $1 \leq step \leq$

$n - w$, and the subsequence $S(a, w)$ is extracted at that offset a. There are a total of $\frac{(n-w)}{step} + 1$ subsequences in T:

$$windows(T, w, step) = \bigcup_{i=0}^{\frac{(n-w)}{step}} S(i \cdot step + 1, w) \tag{2.5}$$

2.2 Time Series Similarity

The *similarity* of two time series Q and C is expressed in terms of a real value using a *distance measure*:

$$D(Q, C) \rightarrow \mathbb{R}_0^+ \tag{2.6}$$

The *similarity measure* is the inverse of the distance measure: it qualifies *similar* time series by a *small* value and *dissimilar* time series by a *large* value. A *distance metric* is a special distance measure that satisfies four axioms.

Definition 1. A *distance metric* is a function that follows four specific axioms:

1. Non-negativity: $D(Q, C) \geq 0$
2. Identity: $D(Q, C) = 0$, *if and only if* $Q = C$
3. Symmetry: $D(Q, C) = D(C, Q)$
4. Triangle inequality: $D(Q, C) \leq D(Q, T) + D(T, C)$ for any time series T,Q, and C.

The fourth axiom is of particular importance as it allows for exact time series indexing. If it is not satisfied, an index structure for time series can only return approximate results to a query.

Three categories of similarity queries

The distance can be determined between two whole time series (*whole matching*). Sometimes it is useful to find the subsequence to a short query within a long time series that minimizes the distance (*sequence to subsequence matching*). For example, when searching for abnormal heartbeats in a long ECG recording. We can further extract characteristic subsequences from the query, and search for each query subsequence for the matches within the long time series (*subsequence to subsequence matching*). Similarity queries can be divided into these three categories:

1. *Whole matching*: given a time series dataset DS of size N and a query Q, all time series have length n, find the time series most similar to Q according to the distance measure D:

$$D_{whole}(Q, DS) = min\{D(Q, T) \,|T \epsilon DS\} \tag{2.7}$$

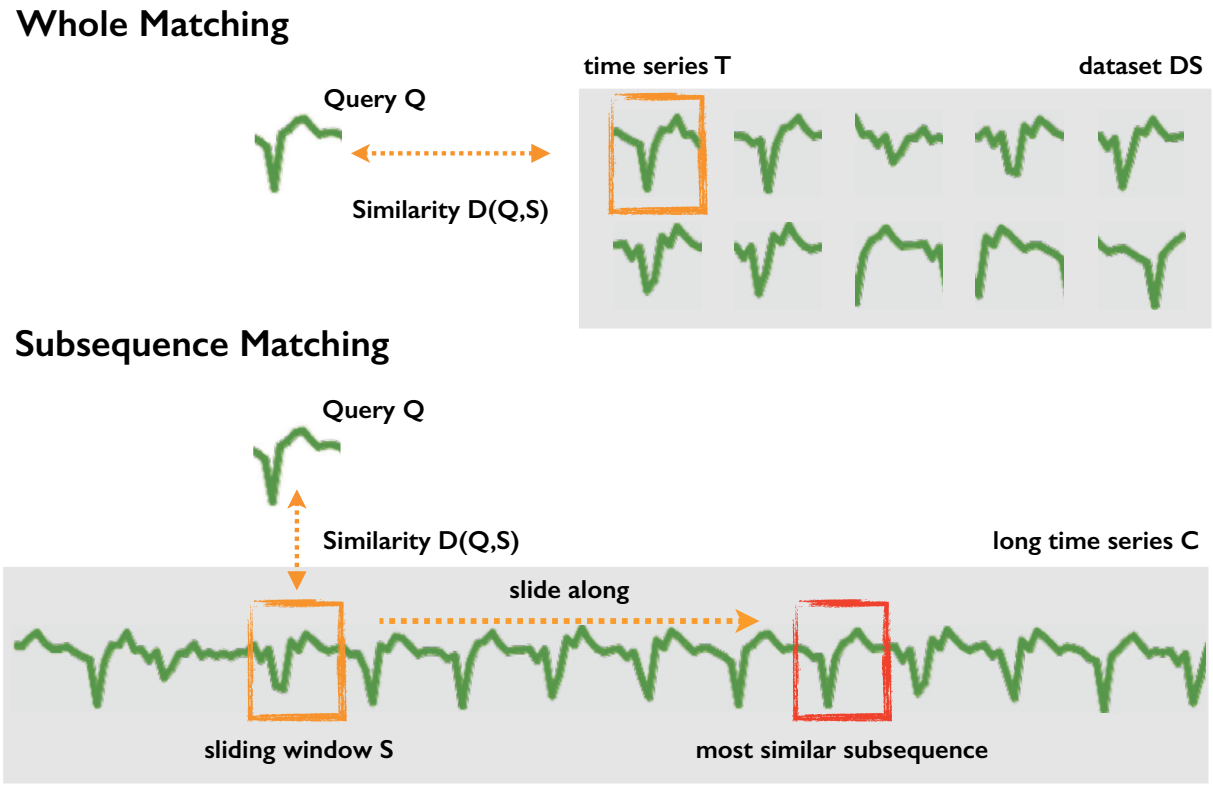


Figure 2.4 – Whole matching (top) and subsequence matching (bottom). Subsequence matching uses a sliding window to extract equal-length subsequences and find the offset that minimizes the distance to the query.

Figure 2.4 (top) illustrates whole matching. Whole matching has a computational complexity of $O(Nn)$ when using the Euclidean Distance.

2. *Sequence to Subsequence matching*: given a short query Q of length w and a long time series C of length n, find the subsequence S within time series C that is most similar to Q:

$$D_{seqsub}(Q, C) = min\{D(Q, S) \,|S\, \epsilon\, windows(C, w, step)\} \tag{2.8}$$

Figure 2.4 (bottom) illustrates subsequence matching. It can be thought of as sliding a window along the time series C and extracting subsequences of length w at each offset. The distance to Q is computed for each extracted subsequence. Sequence to subsequence matching has a computational complexity of $O(w(n-w))$ when using the Euclidean Distance.

3. *Subsequence to Subsequence matching*: given a long query Q of length n and a long time series C of length m. Split Q into c subsequences of length w:

$$windows(Q, w, step) = \{S(1, w), \ldots, S(c, w)\} \tag{2.9}$$

Next, sequence to subsequence matching is applied for each subsequence $S(i, w)$ and C:

$$D_{subsub}(Q, C) = \sum_{S \epsilon windows(Q,w,step)} D_{seqsub}(S, C) \tag{2.10}$$

The subsequences extracted from Q could also be characteristic/frequent, sliding, or disjoint subsequences. Subsequence to subsequence matching has a computational complexity of $O(w(n-w)(m-w))$ when using the ED as the distance measure.

Desirable properties of distance measures

Time series data is typically distorted and either (a) (human) pre-processing can be applied to remove the distortion, or (b) a distance measure can be used that is robust / invariant to these distortions. Several desirable invariances have been identified in the context of time series [14, 30]:

1. *Amplitude / offset:* Time series on different scales will not match well, even if they have a similar shape. For example, a share measured in Euro will not match well to the same share measured in Dollar. A human would convert the stocks to the same currency prior to the comparison.

2. *Local scaling / warping:* This invariance is important in domains in which the duration of a particular event may vary, such as all biological signals. For example, a human may pronounce a single word in a sentence at different speeds. If this invariance is not provided, though the word is the same, the signals will be dissimilar.

3. *Uniform scaling:* Similar to local scaling, uniform scaling deals with the alignment of signals of varying lengths. If this invariance is not provided a song played at twice the speed will not appear to be similar to a song played at normal speed. A human can easily detect the similarity of both songs.

4. *Phase:* This invariance is important when dealing with unaligned signals. For example, when a bird sound is recorded by two different microphones but the recordings start at different time stamps.

5. *Occlusion:* When dealing with missing data, this invariance becomes important. Assume two humans pronounce the sentences: "It is about time" and "It is about *the* time". To make these sentences similar, a distance measure has to be invariant to the absence of the word "*the*".

6. *Noise*: Noise becomes important when dealing with data recorded from sensors. The unwanted noise interferes with the signal and does not add any relevant information. To identify the characteristic shape of a time series, the noise has to be removed.

A useful distance measure is robust to a large subset of these invariances.

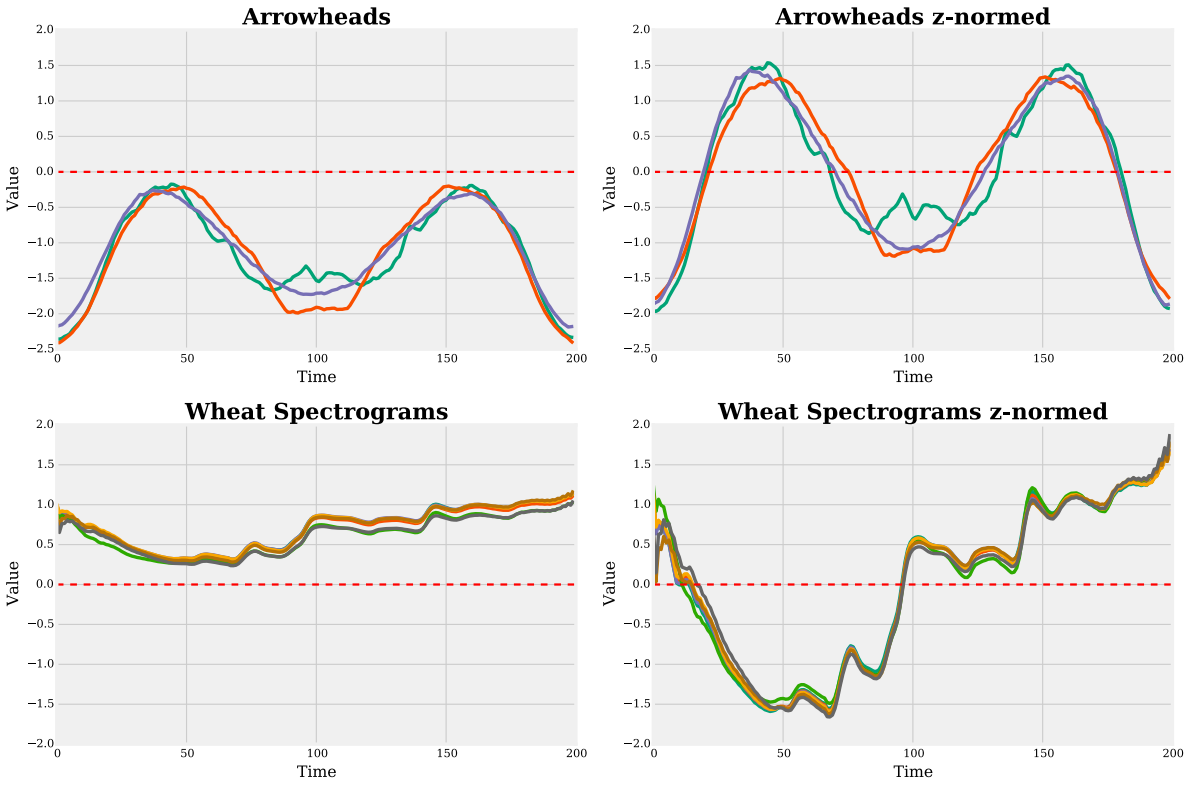


Figure 2.5 – Z-normalization of time series.

Normalization (Amplitude/offset invariance)

A common pre-processing step to obtain offset/amplitude invariance is to apply *z-normalization* to the data. Z-normalization is applied by subtracting the mean μ from a time series to obtain *offset* invariance and by dividing it by the standard deviation σ of the time series to obtain invariance to different amplitudes:

$$z\text{-}norm(T) = (t'_1, \ldots, t'_n) \tag{2.11}$$

$$with$$

$$t' = \frac{t_i - \mu}{\sigma}, \qquad i \, \epsilon \, [1..n] \tag{2.12}$$

Figure 2.5 illustrates arrowhead contours and wheat spectrograms before and after z-normalization. An advantage of this transformation is that it preserves the original features (shape) of a time series.

What are common time series distance measures?

The most common distance measures for time series are Dynamic Time Warping (DTW) or the Euclidean Distance (ED) with a strong consensus that DTW is among the best

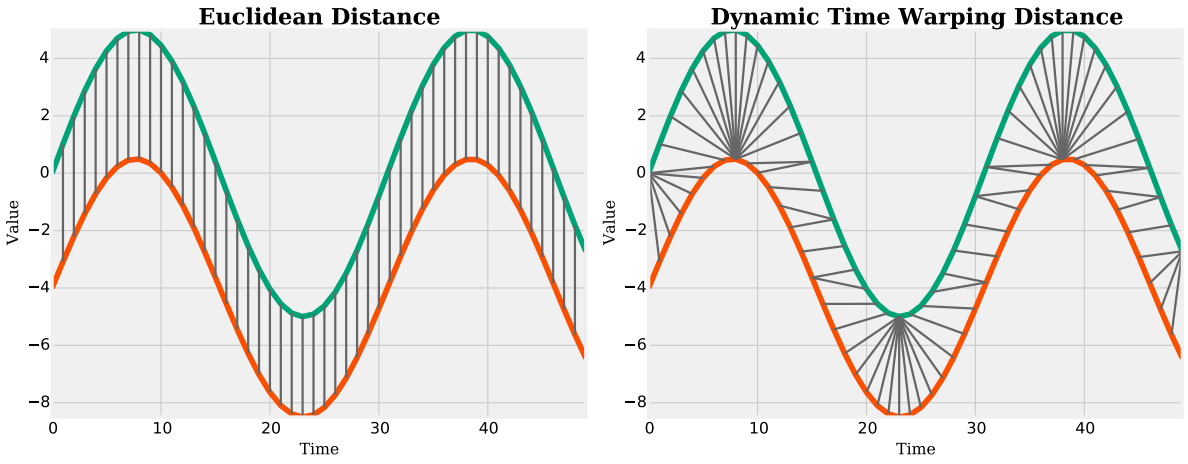


Figure 2.6 – Euclidean Distance and Dynamic Time Warping Distance. The latter applies an elastic transformation to the time axis (time warping).

time series similarity measures [29, 13, 49]. Figure 2.6 illustrates these two distance measures. The ED uses a linear alignment of the time axis. The DTW applies an elastic transformation to the time axis (warping invariance). The ED is a distance metric, and DTW is a distance measure, as it does not satisfy the triangle inequality.

Definition 2. Euclidean distance (ED): The Euclidean distance between two time series $Q = (q_1, \ldots, q_n)$ and $C = (c_1, \ldots, c_n)$, both of length n, is defined as:

$$D_{ED}(Q, C) = \sqrt{\sum_i (q_i - c_i)^2} \tag{2.13}$$

The ED distance metric has some known shortcomings: it does not provide any of the invariances introduced before and it cannot cope with variable length time series [69, 68] (compare Figure 2.2). The ED has a linear computational complexity of $O(n)$.

Dynamic Time Warping [17, 80] deals with distortions in the time axis. It applies an elastic transformation of the time series to detect similar shapes that have a different phase. This is essentially a peak-to-peak and valley-to-valley alignment of two time series. The intuition of DTW is: DTW can be thought of as an extension of the ED, which uses two indices i and j representing both time axis. These indices are incremented independently: $D_{DTW}(Q, C) = \sqrt{\sum_{(i,j)} (q_i - c_j)^2}$.

The DTW algorithms starts by computing a *cost matrix* M that contains the distances between all pairs of values in Q and C (Figure 2.7 top right). This distance matrix has the dimensionality n^2 and is given by:

Definition 3. The *cost matrix* $M \epsilon \mathbb{R}^{n \times n}$ between $Q = (q_1, \ldots, q_n)$ and $C = (c_1, \ldots, c_n)$ is given by:

$$M_{i,j} = (q_i - c_j)^2, \qquad i, j \, \epsilon \, [1 \ldots n] \tag{2.14}$$

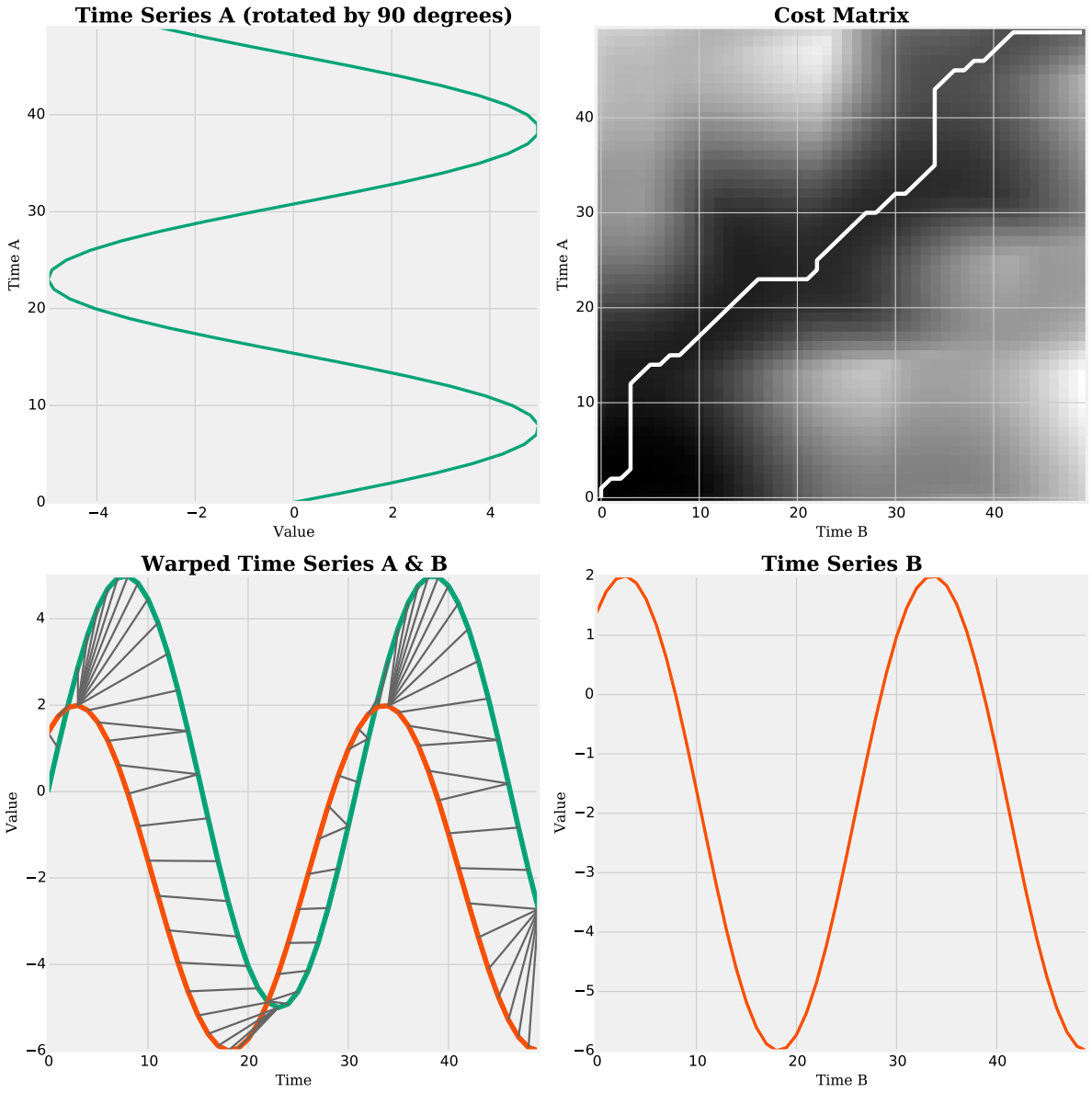


Figure 2.7 – Dynamic Time Warping: cost matrix and optimal warping path (top right) for two time series (top left and bottom right). Bottom left shows the resulting elastic transformation of the time axes.

DTW searches for the optimal warping path. A *warping path* is a traversal of the matrix M.

Definition 4. A *warping path* p in M is defined as a set of tuples that defines a traversal of the cost matrix, whereas i and j represent the indices of the values in $Q = (\underbrace{q_1}_{1=i_1}, \ldots, \underbrace{q_n}_{n=i_n})$

and $C = (\underbrace{c_1}_{1=j_1}, \ldots, \underbrace{c_n}_{n=j_n})$ respectively:

$$p = \{(1,1), (i_2, j_2), \ldots, (i_{n-1}, j_{n-1}), (n,n)\} \tag{2.15}$$

A valid warping path has to satisfy two conditions:

- The start point is $(1,1)$ and the end point is (n,n), and
- The path does not have to proceed but it may not proceed by more than one index: $0 \leq i_{a+1} - i_a \leq 1$ and $0 \leq j_{a+1} - j_a \leq 1$, for all $a < n$.

The ED is a special case of the DTW and is given by the matrix diagonal as the warping path. The DTW distance between two time series is then defined as the warping path through the cost matrix that minimizes the total distance (white line in Figure 2.7 top right and bottom left).

Definition 5. Dynamic Time Warping (DTW) distance: The DTW distance between two time series Q and C is defined as the path p through the cost matrix M with the minimal total distance:

$$D_{DTW}(Q,C) \quad = \quad min \left\{ \sum_{(i,j)\epsilon p} (q_i - c_j)^2 \,|\, p \,\epsilon\, M \right\} \tag{2.16}$$

As DTW is essentially a peak-to-peak and valley-to-valley alignment of two time series, it may fail if there is a variable number of peaks and valleys. For example a variable number of gait cycles in two recordings of human walking motions or an unequal number of heart beats in two ECG recording.

DTW has a quadratic computational complexity of $O(n^2)$. Searching for the optimal path is a time consuming operation. By restricting the amount of warping allowed for each pair of points of the warping path p, the complexity can be reduced significantly. A warping window constraint $r \,\epsilon\, [0.01, 0.02, \ldots, 1]$ is the amount of warping allowed between each pair of points i and j:

$$|i - j| \leq r \cdot n \quad \forall (i,j) \,\epsilon\, p \tag{2.17}$$

DTW with a warping window constraint r has a computational complexity of $O(nr)$. This is referred to as *DTW CV* (Dynamic Time Warping with a warping window constraint set through Cross Validation) in literature.

Categories of Similarity Measures

Similarity measures can be characterized by [48]:

1. *Shape-based*: these are based on the whole time series and perform a point-wise comparison. Shape-based techniques include ED or DTW. Typically, shape-based techniques work well for short time series but fail for noisy or long data.

2. *Structure-based*: the similarity of two time series is based on higher-level structures in the signals. It aims at extracting characteristic subsequences from a time series and searching for similarities in the subsequences of two time series. This can significantly improve the performance [12]. Our Shotgun distance, BOSS model, and BOSS VS model are examples for structure-based similarity measures.

2.3 Time Series Representations / Dimensionality Reduction Techniques

Time series are high-dimensional data: A time series consisting of n measured values can be seen as a point in n-dimensional space, where the i-th measured value represents the i-th dimension. Instead of working directly with the raw and high-dimensional time series data, characteristic features are extracted, and the data of reduced dimensionality are processed. Time series representations are also called *dimensionality reduction techniques*, as their main focus is to significantly reduce the length/dimensionality of the time series. A *time series representation* should provide [30]:

- *Dimensionality reduction*: Reduce the length of the time series, i.e., the data dimensionality, significantly.
- *Noise removal*: Emphasize fundamental shape features while reducing irrelevant features like noise.
- *Speedup*: Has a low computational costs when computing the representation and when comparing two representations.
- *Storage reduction*: Provide some kind of compression.
- *Precision*: Minimize the loss of information, i.e., the reconstruction error.

Many time series representations have been proposed, each one offering different trade-offs. A good dimensionality reduction technique will find representations of the original data, so that each representation is distinct. For example, similar time series will have the same representation after an approximation if the reduced dimensionality is too low and all map to the same point in space.

Time series representations can be divided into two groups: *symbolic* and *numeric*. In these methods a time series is represented as a sequence of discrete values (symbols) or real values respectively. Since symbolic representations are essentially a character string, they can also be used in data structures and algorithms in the field of data-mining such as tries (from re*trie*val), hashing, bag-of-words, or Markov models. Due to their space efficiency they allow for indexing large datasets in main memory [20, 82, 76].

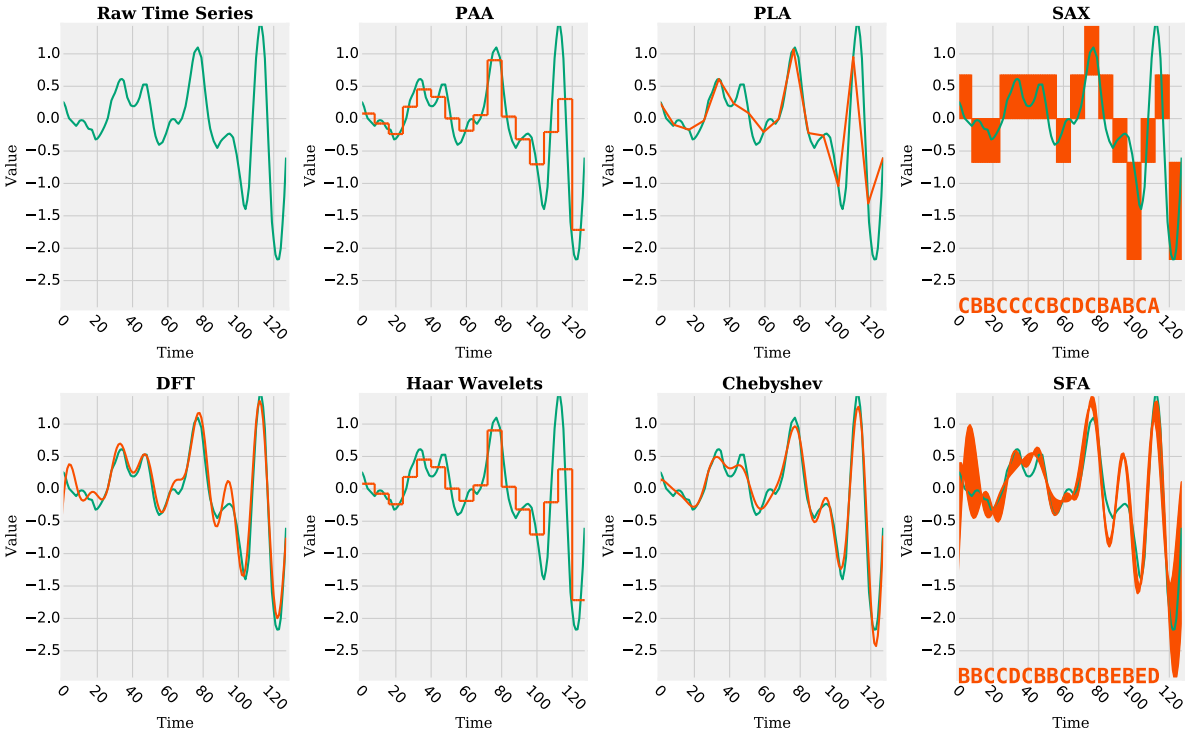


Figure 2.8 – Seven time series representations. Each representation uses 16 values to represent the raw time series of length 128. SAX applies quantization on top of PAA approximation, thus each symbol represents an area. SFA applies quantization on top of the Fourier transform. In concept this is similar to an envelope around the Fourier transform.

Numeric Representations: *Numeric* dimensionality reduction techniques are inspired by signal processing techniques like filters, or approximation/compression techniques [34]. The most common *numeric dimensionality reduction techniques* are (compare Figure 2.8):

- *Discrete Fourier Transform* (DFT) [31, 4] based on the lower Fourier coefficients,
- *Discrete Wavelet Transform* (DWT) [24, 57, 23] based on the lower Wavelet coefficients,
- *Piecewise Aggregate Approximation* (PAA) [41] and *Adaptive Piecewise Constant Approximation* (APCA) [22] based on mean values,
- *Chebyshev Polynomials* (CHEBY) [19] based on the Chebyshev Transformation,
- *Piecewise Linear Approximation* (PLA) [25] based on line segments, or
- *Singular Value Decomposition* (SVD) [44] based on the first Eigenvalues.

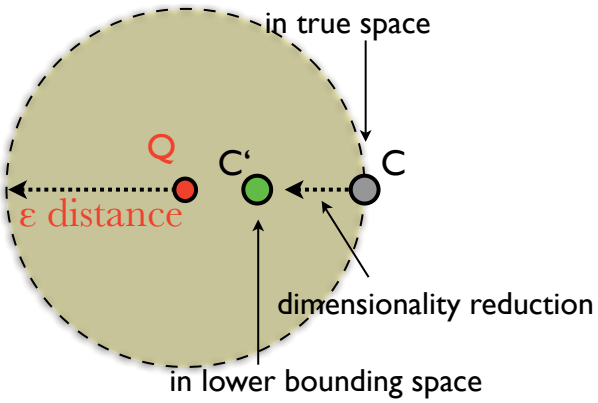


Figure 2.9 – The lower bounding distance has to underestimate the true distance.

Symbolic Representations: The process of transforming a time series into a *symbolic representation* can be generalized to two parts:

1. *Approximation* is applied to map a time series into lower dimensional space resulting in a vector of real values, and

2. *Quantization* is used to map each real value to a discrete value, which can be interpreted as a symbol.

Numeric representations only apply approximation. Symbolic representations add a second level of complexity reduction (noise removal) by introducing the quantization step.

Symbolic Aggregate approXimation (SAX) [46] is the most common symbolic representation. A time series is represented by a sequence of symbols using an alphabet of fixed size for each symbol. SAX is based on PAA for approximation and equi-depth bins of the normal distribution for quantization. *Indexable SAX* (iSAX) [82, 20] is an extension of SAX that introduces a method to dynamically adapt the alphabet size and thereby allows for space-efficient indexing.

The Lower Bounding Lemma

An important property of time series representations is the *lower bounding lemma* [31]. Approximation comes with a loss of information and the distance of any two time series may not be preserved after feature extraction. By providing a distance measure on the approximations $D_{lower}(Q, C)$ that underestimates the true distance $D(Q, C)$ on the two time series Q and C, we can design an algorithm that is guaranteed to be exact. That is, when we search for a query in reduced space using D_{lower} we are guaranteed to find the same results as if the query was evaluated in original space using distance D (aka: no *false dismissals*).

The intuition is simple: If we search within a fixed distance ε around a query in reduced space using D_{lower}, we will retrieve a superset of the actual result. The superset

Figure 2.10 – Similarity Queries: epsilon-range and 5-nearest-neighbor queries. Epsilon range returns the set of times series within distance ε to the query. K-nearest-neighbor returns the set of k time series closest to the query.

is then filtered for false alarms using the true distance D. The result is the same as if we searched in original space.

This property is called the *lower bounding lemma*:

$$D_{lower}(Q, C) \leqslant D(Q, C) \tag{2.18}$$

A lower bounding distance measure has to be defined for each pair of distance measure and time series representation.

It has been established to provide a lower bounding distance measure to the ED for each time series representation. However, as the ED does not provide any invariances to distortions, we believe that this is not always meaningful. Instead, it might be more useful to design a distance measure on a representation that provides a large subset of the invariances to distortions.

2.4 Similarity Search / Query by Content

Similarity search (aka query by content) is at the core of all time series data analytics tasks. It describes the task of retrieving a set of time series that are most similar to a user provided query. The most common similarity queries are nearest-neighbor (NN) and epsilon-range queries. Figure 2.10 illustrates both types of queries.

Definition 6. *Epsilon-range query*: an epsilon-range query is a similarity query in the dataset DS that returns for a query Q the subset $epsilon(Q) \subseteq DS$ that contains all time series within distance $\varepsilon \epsilon \mathbb{R}_0^+$ to the query:

$$epsilon(Q) = \{T \,|T \epsilon DS \wedge D(Q, T) \leq \varepsilon\} \tag{2.19}$$

It can be very difficult to define epsilon without specific knowledge of the data: it is not known how many results a range query might return. If epsilon is to small, no time

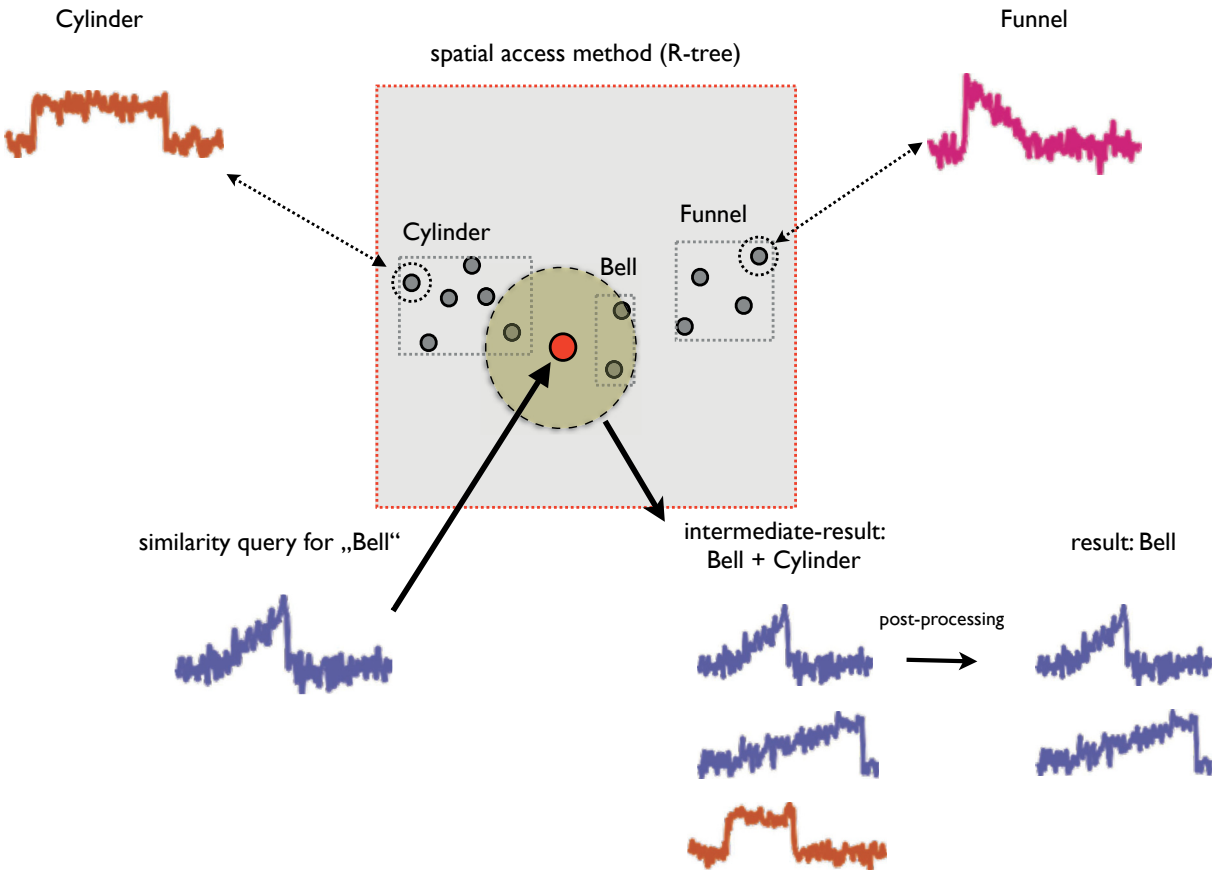


Figure 2.11 – Similarity Search using a Spatial Access Method (SAM).

series are returned. If epsilon is too large, the query can return all time series of a dataset and the execution can be very expensive computationally.

Instead k-nearest-neighbor queries can be used. These always return exactly the k time series with minimal distance (closest) to a query.

Definition 7. *k-nearest-neighbor (k-NN) query*: a k-NN query is a similarity query in the dataset DS which returns for a query Q the subset $nn_k(Q) \subseteq DS$ that contains exactly the k time series that satisfy the condition:

$$\forall C \epsilon nn_k(Q),\ \forall T \epsilon (DS \setminus nn_k(Q)) : \qquad D(C,Q) \leq D(T,Q) \tag{2.20}$$

Another common kind of queries are reverse nearest-neighbor queries [89, 84, 83]. These return for a query Q all those time series T, where Q is in the list of k-nearest-neighbors of T: $Q \varepsilon nn_k(T)$.

Time Series Indexing

The trivial implementation of a similarity search query using the ED has a complexity of $O(Nn)$: a sequential scan of N time series is performed and the distance between the query of length n to each time series is calculated. A problem with this approach is that the amount of time series data can be massive. To avoid a sequential scan for each query, an index structure can be used. The aim of an index structure is to partition the search space into equal-sized groups containing similar entries and allow for a fast lookup for these entries, effectively reducing the complexity to a best case complexity of $O(n \log N)$.

Indexing high dimensional data is a considerable challenge, as spatial access methods (SAMs) [33], like the R-Tree [36, 79, 16] or the KD-tree [18] suffer from a phenomenon called the *Curse of Dimensionality* named by Richard Bellman [17]: with increasing dimensionality of the search space the number of time series in a dataset that need to be examined grows exponentially. As a result the execution time of a similarity query using a SAM can take more time than a sequential scan of all data. SAMs typically degenerate from $10 - 20$ dimensions, equal to time series of the same length. A SAM is also called *multidimensional index structure* or *spatial index*.

Work in [31, 4] (GEMINI) has introduced the idea of dimensionality reduction prior to indexing, and proved that, by using a *lower bounding distance measure*, queries are guaranteed to return the exact same result in reduced dimensional search space as if they were executed in the original search space. By use of a dimensionality reduction technique the Curse of Dimensionality is shifted to 10-20 indexable dimensions of the approximations.

A similarity search for query Q using a lower bounding distance D_{lower} and the true distance D is based on three steps. This method assumes that the SAM and the approximations fit into main memory, while the raw time series are stored on disk:

1. Spatial index lookup: Uses the index and the lower bounding distance D_{lower} to execute the similarity query Q in reduced dimensional space. The lookup ends in the leaf nodes.

2. Disk lookup: Obtains the raw time series from disk. All time series in the leaf nodes build the intermediate result, which is a superset of the actual result.

3. Post-process intermediate result: Filters the raw time series for false alarms using the true distance D.

Figure 2.11 illustrates the similarity search for a Bell curve in 2d-space. The intermediate result contains Bell and Cylinder curves. After post-processing the Bell curves remain.

Typical SAMs used for time series indexing are the R-tree and its variants [36, 31, 4], the TS-tree [10], or the iSAX 2.0 index [82, 20].

A problem with SAMs is that the length of the query has to be known ahead of time to create the index. That means it is not trivial to answer arbitrary query lengths using

a SAM, built from a fixed time series length. This is due to the pre-processing (like z-normalization) of the time series [62].

Approximate Similarity Search

GEMINI is a framework for *exact similarity search*. Due to the limitations caused by the Curse of Dimensionality, *approximate similarity search* has become popular. That is, a similarity search is not guaranteed to return the true nearest neighbor to a query (allows for false dismissals). Approximate similarity search is a trade off between accuracy and fast execution times. One of the most popular representatives of approximate similarity search is *locality sensitive hashing (LSH)* [8, 15, 27]. The key idea is to use a family of locality-sensitive hash functions to map each high-dimensional object to a label. Locality sensitive hashing implies that if two time series are similar according to a distance measure, these should hash to the same label, called bucket. Dissimilar time series should map to different labels. To perform an approximate similarity search, the query has to be hashed and all time series from the same bucket are returned as candidates. Finally, the time series in the set of candidates are ranked according to their distance to the query using the distance measure. By the use of multiple hash tables, the quality of the candidate set can be improved significantly. Typically, there are several hundred hash tables. LSH Forest [15] is an index structure for similarity search that (a) builds a prefix tree (LSH tree) based on the prefixes of the bucket labels, and (b) builds an ensemble of LSH trees. The LSH tree eliminates the need for data-dependent parameter tuning.

2.5 Time Series Data Analytics

Data analytics describes the task of discovering novel, interesting and useful information by applying algorithms to extract hidden knowledge from raw and potentially large time series data. As described above, at the core of all time series data analysis techniques there are a *time series representati*on and a *similarity measure* to compare two time series. Data analytics tasks can be divided whether (a) they are applied to whole time series or (b) whether subsequences of a long time series are analyzed. Typical areas of research include classification, clustering, motif discovery, and discord discovery (anomaly detection) [30, 37] (see Figure 2.12).

Classification [7]: Classification describes the task of predicting a class label for a time series, whose label is unknown. A classifier has to produce a model from a set of labeled time series that takes an unlabeled time series as input and outputs its label. As the labels have to be known beforehand, this is referred to as supervised learning. Classification algorithms include k-nearest-neighbor classifiers, decision trees, SVMs, Hidden Markov Models (HMMs), Artificial Neural Networks, to name but a few examples.

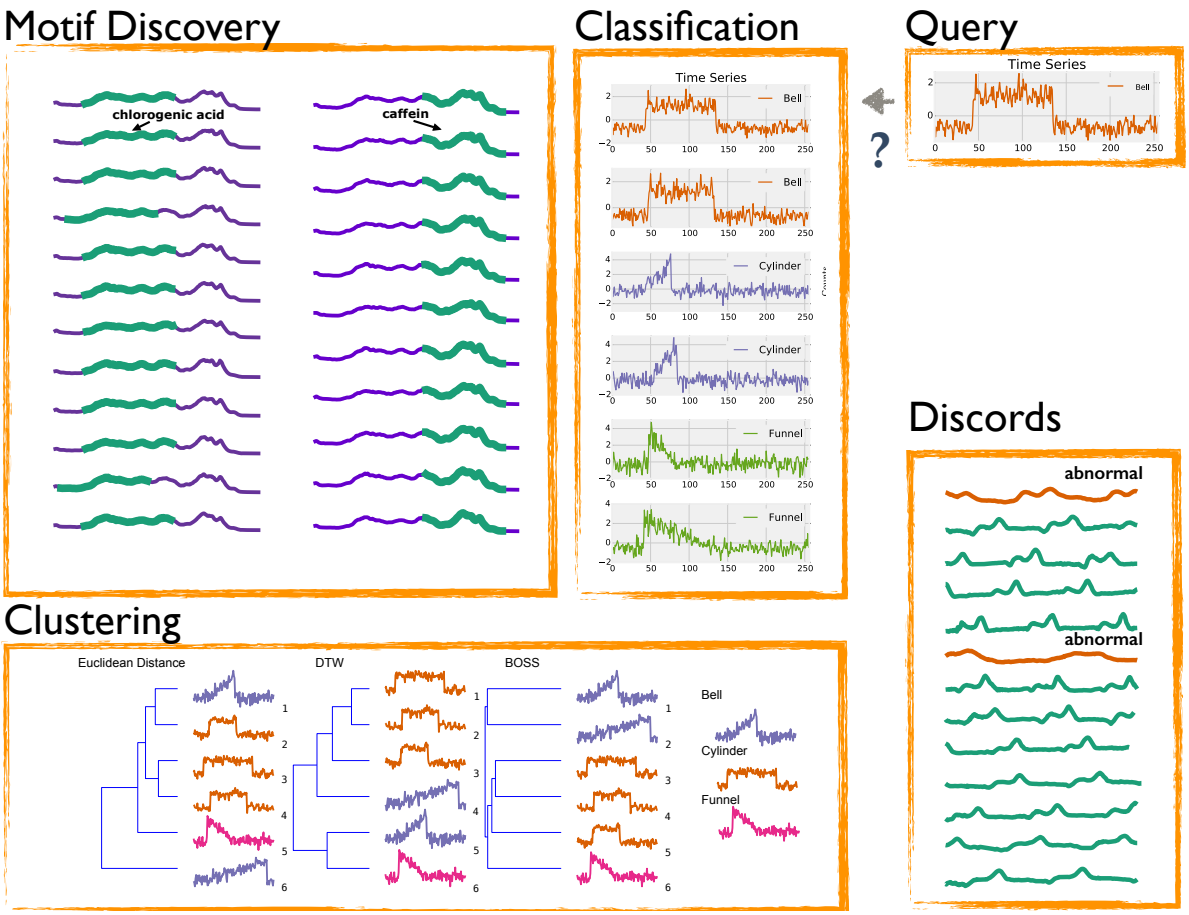


Figure 2.12 – Time Series Data Analytics.

Clustering [94, 88]: Clustering aims at finding clusters of similar data in a dataset. This is achieved by maximizing inter-cluster similarity while minimizing intra-cluster similarity. The main difference to classification is that the labels are not known in advance and are obtained as a result of the clustering (unsupervised learning). It can be divided into: whole time series clustering and subsequence clustering. Subsequence clustering aims at discovering interesting/common patterns in one time series, like heartbeats in a long ECG recording. Clustering algorithms can be divided into:

- Hierarchical clustering: A hierarchy/tree of similar objects based on the pairwise distances is constructed,
- Partitional clustering: Constructs exactly k partitions of the data (k-means, k-medoids),
- Density-based clustering: Grows the clusters from seeds as long as the density exceeds a threshold (DBSCAN).

Motif Discovery: The aim is to find frequently occurring subsequences within one longer time series or across multiple time series. Motif discovery can be used as a subroutine for classification or clustering algorithms. First, motifs are extracted and then these motifs are used as an input for higher-level analysis using classification/clustering algorithms.

Discord Discovery [35]: As opposed to motif discovery, discord detection aims at searching abnormal, thus rare, subsequences. To detect outliers, a model of normal behavior has to be learned first. The subsequence that is maximally different from all remaining subsequences in a time series is called a discord. Discords are useful in wildlife/bird monitoring, intrusion detection, data cleaning, or anomaly detection, for example.

Prediction: Prediction aims at forecasting new values of a long time series. Given an input time series with a periodical (predictable) structure, the goal is to forecast upcoming values within a future timespan.

Machine Learning Techniques for Time Series Analysis: While classical machine learning algorithms (SVM, Random Forest, Decision Tree, ...) are very common in areas of research such as audio or image processing, there has been no major breakthrough in time series analysis, yet[1]. This observation has since been confirmed in several experimental studies [13, 29, 68, 12].

As a result, Keogh et al. and Bagnall et al. claim that DTW is among the best similarity measures and should be used as the benchmark to compare to in the context of time series data analytics [13, 29, 12].

2.6 Summary

Time series databases result from recording data over time. Time series may be recorded at variable lengths, and can be erroneous, extraneous due to noise, dropouts, or subtle distinctions, and can be highly redundant due to repetitive (sub-)structures. Time series databases are analyzed based on similarity. The *similarity* of two time series is expressed in terms of a real value using a *distance measure.* A *distance metric* is a distance measure that satisfies four specific axioms and allows for efficient time series indexing. The *similarity measure* is the inverse of the distance measure: it qualifies *similar* time series by a *small* value and *dissimilar* time series by a *large* value. Unlike exact search, similarity based search finds results that are similar to a query based on a similarity measure.

[1] "However the unique structure of time series means that most classic machine learning algorithms to not work well for time series. In particular the high dimensionality, very high feature correlation, and the (typically) large amounts noise that characterize time series data have been viewed as an interesting research challenge". Keogh et al. [43]

Working with time series is difficult as the definition of similarity depends on the application domain and the data analytics task. Furthermore the time series data may be distorted and constitute large databases. A human can easily abstract from distortions of a signal and extract a general model (the shape) from the time series. Based on this model, similarity can be determined.

At the core of each data mining technique there are a *representation (the data model)* of the time series and a *similarity measure* to compare two time series. The difficulties arise from defining a representation that maintains the characteristic features of the time series, such as a human would, and defining of a similarity measure based on human perception.

Time series representations aim at dimensionality reduction, noise removal, speedup, and storage reduction. Representations can be divided into two groups: symbolic and numeric. In these methods a time series is represented as a sequence of discrete values (symbols) or real values respectively.

The most common *distance measures* for time series are Dynamic Time Warping (DTW) or the Euclidean Distance (ED) with a consensus that DTW is among the best time series similarity measures. ED or DTW do only work well for short time series but typically produce meaningless results for noisy or long time series data. Thus, the data has to be pre-processed first.

Indexing high dimensional data is a considerable challenge, as spatial access methods (SAMs) suffer from a phenomenon called the *Curse of Dimensionality*. A framework named GEMINI has introduced the idea of dimensionality reduction prior to indexing, and has proven that, by using a *lower bounding distance measure*, queries are guaranteed to return the exact same result in reduced dimensional search space as if they were executed in the original search space.

Typical areas of time series research include classification, clustering, motif discovery, and discord discovery.

Chapter 3

Symbolic Fourier Approximation: A Symbolic Representation of Time Series and an Index for High Dimensional Datasets

The Symbolic Fourier Approximation (SFA) is a time series representation. This chapter gives a detailed description of SFA and the SFA trie. The chapter is based on and contains text passages from my publications [76, 68]. The idea of SFA and the SFA trie were first introduced in my Diploma Thesis at the Free University of Berlin [67].

3.1 Introduction

A time series, consisting of n measured values, can be seen as a point in n-dimensional space, where the i-th measured value represents the i-th dimension. Dimensionality reduction aims at representing a time series by a lower dimensional representation, while retaining the significant features in the reduced representation. These can be divided into two groups: *symbolic* and *numeric* (compare Chapter 2.3). In these methods a time series is represented as a sequence of discrete values (symbols) or real values respectively. Numeric representations apply noise reduction through signal processing techniques like approximation/filtering/compression. Symbolic representations add a second level of complexity reduction that has a noise reducing effect by introducing a quantization step. In this chapter we present our symbolic representation of time series *Symbolic Fourier Approximation (SFA)* that represents each real valued time series by a sequence of symbols, named SFA word, using a finite alphabet of symbols. SFA is composed of approximation using the Fourier transform and quantization using a technique called *Multiple Coefficient Binning (MCB)*. Since symbolic representations are essentially a character string, they can be used in combination with data structures and algorithms in the field of data-mining such as indexing using tries (comes from re*trie*val [32]), bag-of-words, Markov models, or string-matching [47].

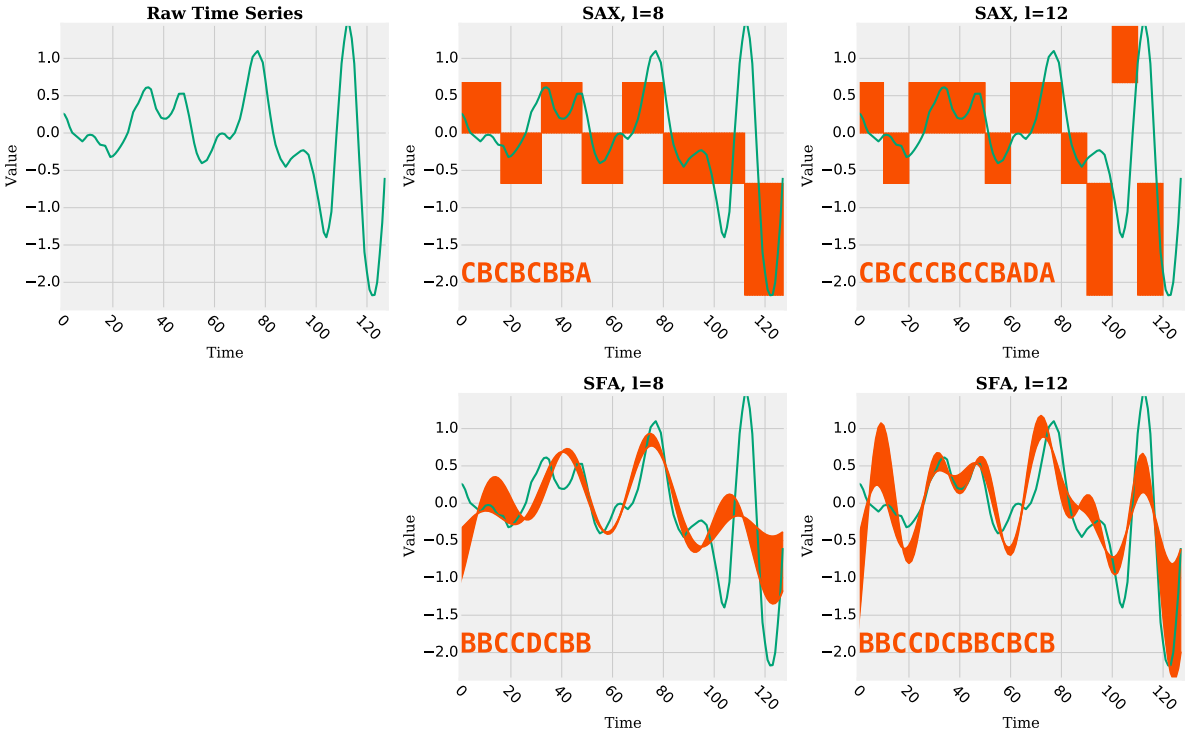


Figure 3.1 – For SAX (top), we (a) have to recalculate all symbols (mean values), or (b) drop the rear part of the time series when changing the word length l. For SFA (bottom), the symbols (Fourier coefficients) of a smaller word length are always a prefix of the larger word lengths.

SFA is a symbolic representation of time series like Symbolic Aggregate approXimation (SAX) [47]. Unlike SAX, which uses mean values (PAA) to approximate a time series, SFA uses DFT coefficients. Both, have a noise canceling effect by smoothening a time series. One disadvantage of using mean values is that these have to be recalculated when changing the resolution - i.e., from weekly to monthly mean values. The resolution of SFA can be incrementally adapted by choosing an arbitrary subset of Fourier coefficients without recalculating the DFT of a time series (compare Figure 3.1).

It is this frequency domain nature of SFA that makes it unique under the symbolic time series representations. Dynamically adding or removing Fourier coefficients to adapt the degree of approximation without recalculating the Fourier transform is at the core of all presented algorithms in this thesis including time series indexing.

Indexing high dimensional time series data is a considerable challenge, as spatial index structures, like the R-Tree [16, 36], suffer from the *Curse of Dimensionality*: with increasing dimensionality of the search space, the performance of similarity based queries on the index becomes worse than a linear scan of all data. Spatial index structures usually degenerate from 10 to 20 dimensions [33].

Work in [31, 4] has introduced the idea of dimensionality reduction prior to indexing,

and has proven that, by using a *lower bounding distance measure* (see Chapter 2.3), queries are guaranteed to return the exact same result in reduced dimensional search space as if they were executed in the original space. In order to reduce the dimensionality of a time series a *dimensionality reduction technique* is applied first (see Chapter 2.3), effectively reducing dimensionality of the time series by 10:1 to 50:1 by approximation. By the use of a dimensionality reduction technique the Curse of Dimensionality is shifted to 10 to 20 indexable dimensions of the reduced space. The problem with dimensionality reduction is that information from the raw time series is lost. A dimensionality reduction technique should find distinct representations of the original data. For example, similar time series will have the same representation in reduced space if the dimensionality was chosen too low. For two reasons the choice of the optimal dimensionality is difficult: (a) with a high dimensionality we overrepresent dissimilar time series, while (b) with a too low dimensionality we underrepresent similar time series, which makes them impossible to distinguish. Both of these issues negatively impact the index up to a point where a similarity search query results in a sequential scan of the whole database, making the index structure superfluous.

We propose a spatial access method (SAM) named *SFA trie* that makes use of a variable number of dimensions for indexing time series in reduced space in order to postpone the impact of the two issues mentioned before. The idea is to group similar time series based on a small common prefix. The length of the prefix is increased until each time series has a distinct representation in reduced space. This can be implemented by using a trie, which is built over a set of strings. This introduces the problem of how to represent a time series as a string. Furthermore, it must be possible to adapt the length of a time series representation on the fly without the need to recalculate it. This is why we use our symbolic representation based on the frequency domain as opposed to the spatial domain. In the frequency domain each dimension contains approximate information about the whole time series. By increasing the length of the representation we can add detail, thus improving the overall quality. In the spatial domain we have to decide on a length of a representation in advance and a prefix of this length would only represent a subsequence of the time series.

In this chapter we introduce a novel symbolic representation called *Symbolic Fourier Approximation* (SFA) and the *SFA trie*, an index structure utilizing the properties of the frequency domain nature of SFA. As part of this technique we:

- Propose the symbolic representation SFA based on the *Fourier transform* for approximation and underline the benefits compared to other dimensionality reduction techniques (Chapter 3.3),
- Introduce a quantization technique which we call *multiple coefficient binning* (MCB) that is a consequence of the use of the Fourier transform where each Fourier coefficient has its own value distribution. MCB helps to improve the pruning of the search space during query execution (Chapter 3.3),

- Provide a proof of a *Euclidean lower bounding distance measure* for SFA which guarantees that the query results of a similarity search in SFA reduced space are the same as in the original space when using the Euclidean distance (Chapter 3.3),
- Introduce the *SFA trie*, based on a prefix tree built from the strings of the SFA words (Chapter 3.4),
- Present a bulk loading algorithm to build the SFA trie for a large amount of time series with limited main memory (Chapter 3.4.3), and
- Show through experiments that SFA and the SFA trie scale to a factor of 5–10 higher indexed dimensions without degeneration than previous approaches and can index terabyte-sized datasets using commodity hardware. Furthermore, the SFA trie is better than state of the art in terms of exact search performance on real and synthetic time series datasets (Chapter 3.5).

3.2 Background and Related Work

We focus on dimensionality reduction techniques that provide a lower bounding distance measure to the Euclidean distance (ED) (Figure 3.2). These allow for efficient similarity search using the GEMINI framework (compare Chapter 3.4).

Numeric dimensionality reductions include Discrete Fourier Transform (DFT) [31, 4, 60], Discrete Wavelet Transform (DWT) [24, 57, 23], Piecewise Aggregate Approximation (PAA) [41], Adaptive Piecewise Constant Approximation (APCA) [22], Chebyshev Polynomials (CHEBY) [19], Piecewise Linear Approximation (PLA) [25], or Singular Value Decomposition (SVD) [44]. With the exception of SVD, excluded because of its unreasonably high cubic computational complexity, we use all of these dimensionality reductions in our experimental evaluation in Chapter 3.5.

Symbolic Aggregate approXimation (SAX) [46] was the first symbolic representation to introduce a Euclidean lower bounding distance measure. A time series is represented by a sequence of symbols using an alphabet of fixed size for each symbol. Our SFA is a symbolic representation of time series like Symbolic Aggregate approXimation (SAX) [47]. Unlike SAX, which uses mean values (PAA) to approximate a time series, SFA uses DFT coefficients. Both, have a noise canceling effect by smoothing the time series. Figure 3.2 shows the differences in concepts. SFA builds an envelope around the Fourier transform of the signal. The SAX representation envelops the mean values over a time series with rectangular shapes.

One disadvantage of using mean values is that these have to be recalculated when changing the resolution - i.e., from weekly to monthly mean values. The resolution of DFT can be incrementally adapted by choosing an arbitrary subset of Fourier coefficients without recalculating the DFT of a time series. Dropping the rear mean values of a SAX word is equal to dropping the rear part of a time series. I.e., the last SAX symbols in

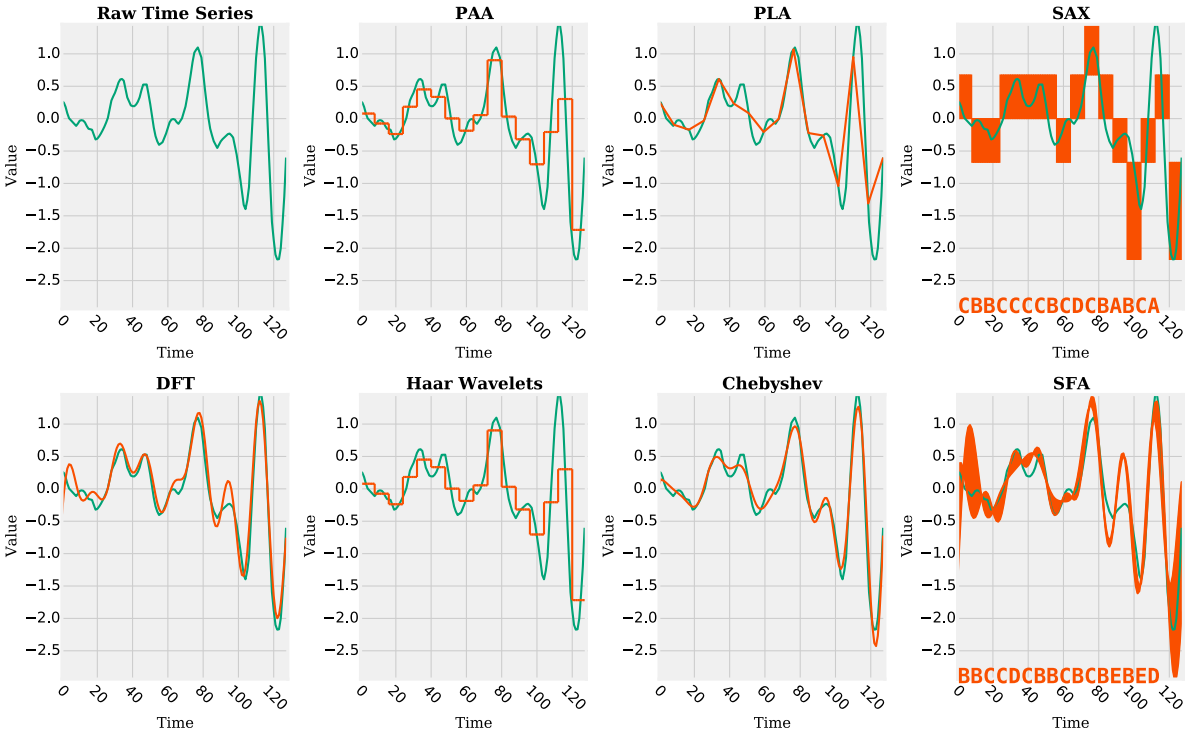


Figure 3.2 – A raw time series and seven different dimensionality reduction techniques. Each one uses 16 features to represent the raw time series of length 128. SFA and SAX are symbolic representations, i.e., a time series is represented by a sequence of symbols. In concept, SFA builds a tight envelope around the Fourier transform of the time series.

Figure 3.2 represent the tail of the time series. Dropping the latter halve of the symbols *CDCBABCA* is equal to dropping the tail of the time series. To avoid this, we would have to recalculate all SAX representations each time we choose to represent a time series by a different SAX word length.

Indexable SAX (iSAX) [82, 20] is an extension of SAX, which allows for efficient indexing through the iSAX index. iSAX introduces a dynamic alphabet size. For each new level in the index, the alphabet size is doubled for one mean value. iSAX 2.0 [20] introduces a novel splitting policy. iSAX 2.0+ [21] introduces a bulk insertion algorithm. The iSAX index and our SFA trie are complementary approaches, as the SFA trie uses a fixed-size alphabet and dynamically increases the length of the SFA words for splitting, while the iSAX index follows the opposite approach: fixed word length and dynamic alphabet size.

Space partitioning index structures like the R-tree [36, 64], KD-tree [18], or TS-tree [10] apply partitioning to k-dimensional space to organize points. The search space is fixed to a constant dimensionality, thus all time series are transformed using this low dimensionality, and the space is split to a finer granularity at each new level of the index. This is opposed to the SFA trie where each approximation is computed at a high dimensionality, but a

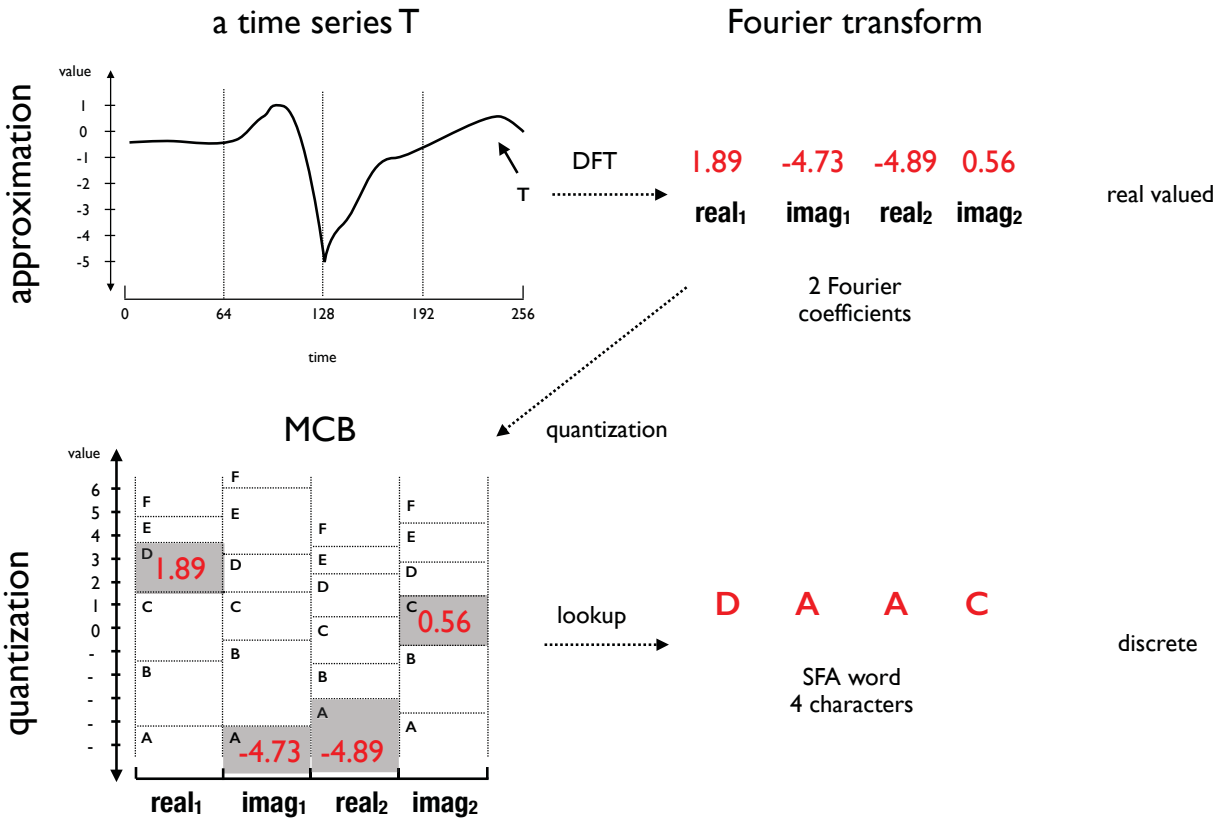


Figure 3.3 – SFA: A time series is (a) approximated (low-pass filtered) using the Fourier transform and (b) quantized using MCB, resulting in the SFA word $DAAC$.

dynamic prefix of each SFA word is used for splitting at each new level of the index.

3.3 SFA: A Symbolic Representation

3.3.1 Motivation: From Real Values to Words

The *Symbolic Fourier Approximation* (SFA) [76] is a symbolic representation of time series. A real valued time series is represented by a sequence of symbols, named *SFA word*, using a finite alphabet of symbols. Numeric dimensionality reduction techniques only use approximation. Symbolic representations add a second level of complexity reduction (noise removal) by introducing a quantization step after the approximation step.

Our symbolic representation SFA aims at (compare Chapter 1.1.1):

- **Noise removal:** Rapidly changing sections of a signal are often associated with noise. These can be removed by the use of a low-pass filter. The SFA word length determines the number of Fourier coefficients and thereby the bandwidth of the low-pass filter.

- **String representation:** Using a string representation allows for string (matching) algorithms like the *prefix tree*, or the bag of words to be used in the field of time series analysis. The size of the quantization alphabet determines the degree of quantization and has an additional noise reducing effect.
- **Frequency domain:** The SFA word length can be dynamically adapted by choosing an arbitrary subset of Fourier coefficients without recalculating the Fourier transform of the time series. Adding Fourier coefficients to an SFA word adds details and reduces the reconstruction error between the transformed and the original time series. This is exploited as part of the SFA trie in terms of variable length index construction, or to train a classification model for the Bag-of-SFA-Symbols model (Chapter 5).
- **Dimensionality reduction:** Similarity search in data with increasing dimensionality results in an exponential growth of the search space, referred to as the Curse of Dimensionality. A common approach to postpone this effect is to apply dimensionality reduction to the original data prior to indexing. Our index structure SFA trie exploits the frequency domain nature of SFA as it grows in depth using variable prefix lengths of the SFA words for indexing.
- **Storage reduction:** SFA has an up to 1000:1 lower memory footprint than the original time series. By the use of
  - "Approximation": The length of the time series is reduced by up to 100:1, and
  - "Quantization": Only a few bits are needed to encode every symbol as opposed to a 8 byte double for real-valued representations, leading to another reduction in size of 8:1 up to 32:1. This allows for indexing terabyte-sized datasets using commodity hardware in main memory.

3.3.2 The Symbolic Fourier Approximation (SFA)

The symbolic representation SFA has two parameters:

- The SFA **word length** $l \epsilon \mathbb{N}$ represents the number of Fourier coefficients for approximation. Commonly, the first Fourier coefficients are used. A smaller SFA word length correlates to a stronger noise reduction by using less Fourier coefficients.
- The SFA **alphabet size** $c \epsilon \mathbb{N}$ is used for quantization. A smaller alphabet size results in a stronger noise reduction and a larger envelope around the Fourier transformed signal.

SFA consists of two phases: a *pre-processing phase* and the *transformation phase* (Figure 3.4). A pre-processing phase is required, as SFA is based on data adaptive quantization intervals. These data adaptive quantization intervals are then used to transform a time

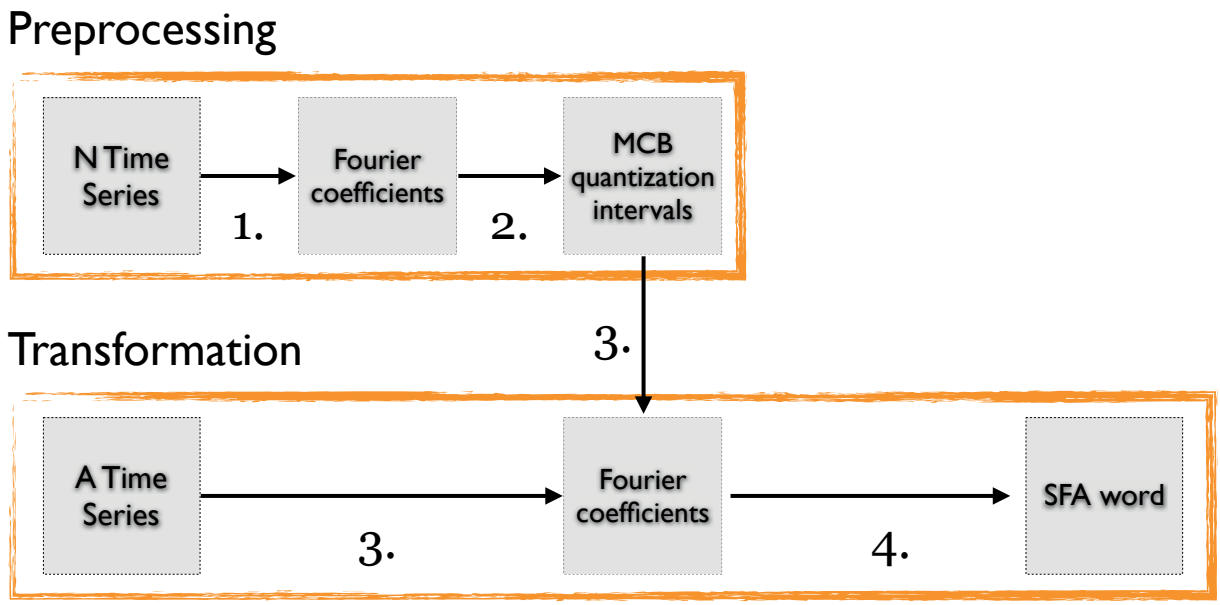


Figure 3.4 – The SFA transformation consists of two phases: pre-processing and transformation.

series to an SFA word. SFA requires a representative train dataset to learn these quantization intervals using our *multiple coefficient binning* (MCB) quantization technique.

1. SFA pre-processing: Determine MCB quantization intervals.

   Input: A set of representative time series.

   (a) **Quantization Training:** All train time series are transformed using the Fourier transform. MCB quantization intervals are trained from the real and imaginary parts of the Fourier coefficients using a quantization technique we call *multiple coefficient binning* (MCB).

   Output: MCB quantization intervals.

2. SFA transformation: Apply MCB quantization to the Fourier coefficients.

   Input: One real-valued time series, the MCB quantization intervals.

   (a) **Approximation:** The time series is transformed using the Fourier transform.

   (b) **Quantization:** The MCB quantization intervals are applied to the real and imaginary parts of the Fourier coefficients.

   Output: An SFA word.

The *approximation step* aims at representing a time series of length n by a transformed signal of reduced length l. Higher order Fourier coefficients represent rapid changes like dropouts or noise in a signal. The signal is low-pass filtered by using the first $\frac{l}{2} \ll n$ Fourier coefficients. Our basic rationale for using the Fourier transform for approximation is simple: The Fourier transform has the lowest reconstruction error when compared to the other dimensionality reduction techniques (see Chapter 3.5.1.2). As a result it offers the tightest lower bounding distance to the Euclidean distance. This is correlated to a reduction of false alarms when performing similarity search in lower dimensional space using GEMINI (compare Chapter 2.4). An advantage of the use of the Fourier frequency domain is that the length of an SFA word can be incrementally adapted by choosing an arbitrary subset of Fourier coefficients without recalculating the Fourier transform of a time series. This allows for variable length index construction (Algorithm 5.3) and can further be used to train a classification model (Chapter 5). Figure 3.3 (top right) illustrates the approximation step. A time series is decomposed using the Fourier transform to its first two Fourier coefficients equal to the values $(1.89, -4.73, -4.89, 0.56)$.

The *quantization step* adds to the noise reduction by dividing the frequency domain into frequency bins (intervals) and mapping each Fourier coefficient to its bin. In essence, the MCB quantization builds an envelope around the Fourier transform of the time series (compare Figure 3.2). The MCB quantization determines equi-depth bins to map the real and imaginary part of the Fourier coefficients *separately* to symbols. As part of MCB a separate histogram for each real and imaginary part is built using all train samples. These histograms are then partitioned using equi-depth binning. Figure 3.3 (bottom right) illustrates the SFA transformation. The two Fourier coefficients $(1.89, -4.73, -4.89, 0.56)$ are quantized to an SFA word $DAAC$ using the MCB quantization bins.

MCB as a quantization technique is not limited to a single dimensionality reduction technique like the Fourier transform. We use the Fourier transform since it shows the best results in our experiments in Chapter 3.5.1.2. A result of the usage of the Fourier transform, which introduces frequency domain, is that each frequency has its own value distribution. This leads to the necessity to apply individual equi-depth quantization to each group of frequencies (see Experiment 3.5.1.2).

3.3.3 Approximation

The idea of the Fourier transform is to decompose a time series into a sum of (orthogonal) basis functions. These basis functions are sinusoid curves. The first few basis functions correspond to slowly changing sections and represent the coarse distribution of a signal, while later basis functions represent rapid changes like gaps or noise in a signal. Thus, the use of only the first few basis functions produces a good approximation of the shape of a time series. The use of the first Fourier coefficients is equal to a low-pass filter and as such has a noise canceling effect by smoothening a time series.

The Discrete Fourier Transform (DFT) is an algorithm used to decompose a *whole time series* into Fourier coefficients. When dealing with *subsequences* extracted from a

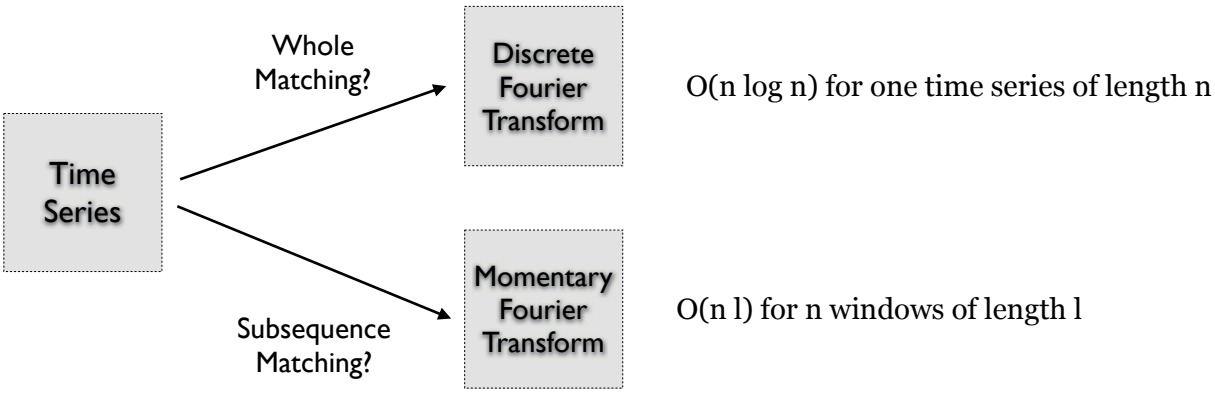


Figure 3.5 – Fourier transform: DFT for Whole Matching and MFT for Subsequence Matching (compare Chapter 2.2).

long time series, there is a significant overlap in computations between each overlapping window (compare Figure 2.3). We use the Momentary Fourier Transform to significantly reduce this computational complexity to linear in the length of a time series (Figure 3.5).

The Discrete Fourier Transform: The *Discrete Fourier Transform* (DFT) is an algorithm to decompose a signal T of length n into a sum of orthogonal basis functions using sinusoid waves. Each wave is represented by a complex number:

$$X_u = (real_u, imag_u), \quad \text{for } u = 0, 1 \ldots, n-1 \tag{3.1}$$

It is called a Fourier coefficient. The n-point DFT of a discrete signal of one variable $T(x)$, $x = 0, 1, \ldots, n-1$, is given by the equation:

$$DFT(T) = X_0, \ldots, X_{n-1} = (real_0, imag_0, \ldots, real_{n-1}, imag_{n-1}) \tag{3.2}$$

with

$$X_u = \tfrac{1}{n} \textstyle\sum_{x=0}^{n-1} T(x) \cdot e^{-j2\pi ux/n}, \quad \text{for } u \,\epsilon\, [0, n)\,, \; j = \sqrt{-1} \tag{3.3}$$

The first Fourier coefficients correlate to lower frequency ranges or the slowly changing sections of a signal. The higher order coefficients correlate to higher frequency ranges or rapidly changing sections of a signal. The first Fourier coefficients are sufficient to describe most signals, thereby applying a low-pass filter. The first Fourier coefficient

$$X_0 = \frac{1}{n} \sum_{x=0}^{n-1} T(x) \cdot e^0 \tag{3.4}$$

is equal to the mean value of a signal and can be discarded to obtain offset invariance (vertical shifts).

A Fourier transformation of a real-valued time series of length n can be represented using $\frac{n}{2}$ Fourier coefficients, as the later Fourier coefficients are *complex conjugates* of the

first $\frac{n}{2}$ coefficients [61], except for the very first Fourier coefficient X_0:

$$\begin{aligned} DFT(T) &= X_0 \dots X_{n-1} && (3.5) \\ &= X_0 X_1 \dots X_{\frac{n}{2}} X^*_{\frac{n}{2}} \dots X^*_1 && (3.6) \end{aligned}$$

with X^*_i is the complex conjugate of X_i.

Computational Complexity: The DFT has a computational complexity of $O(n \log n)$ for time series length n. When transforming N time series of length n the computational complexity is:

$$T(FT_{whole}) \, \epsilon \, O(N \cdot n \log n) \tag{3.7}$$

Momentary (Incremental) Fourier Transform: The SFA transformation is dominated by the computational complexity of a single DFT. For subsequence matching $n - w + 1$ sliding windows of length w can be extracted from a time series of length n. A single DFT of a sliding window of length w has the computational complexity of $O(w \log w)$, resulting in a computational complexity of $O(n \cdot w \log w)$ for all windows. This is clearly too time consuming considering we need only the first $\frac{l}{2} \ll w$ Fourier coefficients ($\frac{l}{2}$ real and $\frac{l}{2}$ imaginary values) for an SFA word of length l. The *Momentary Fourier Transform* (MFT) [6] is an alternative algorithm to calculate the Fourier transform that deals with the significant overlap in computations between the Fourier transform of overlapping windows.

Let us assume, we are interested in the first $\frac{l}{2} \ll w$ Fourier coefficients of the sliding windows $\{S(1, w), \dots, S(n - w + 1, w)\}$ (compare Equation 2.3). A sliding window at time interval i can be inferred from its predecessor by one summation and one subtraction:

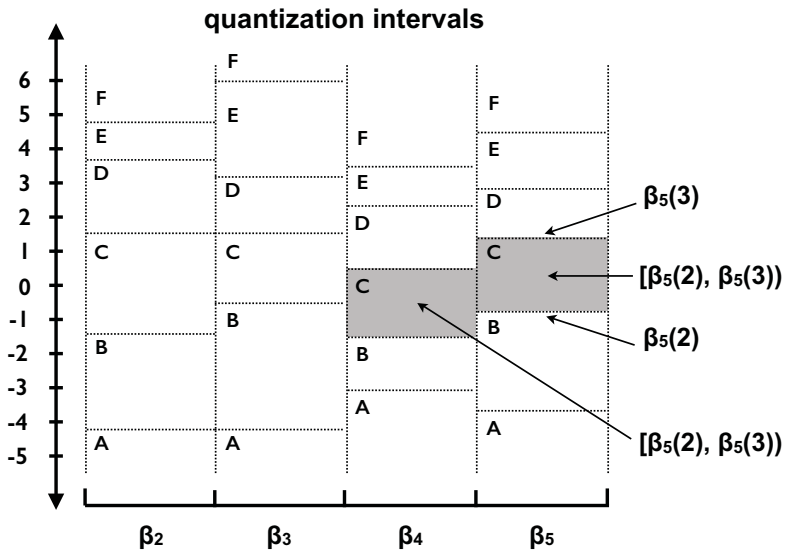
$$S(i, w) = S(i - 1, w) + x_i - x_{i-w} \ , \text{ for } i > 1 \tag{3.8}$$

The MFT makes use of this recursive property as the first $\frac{l}{2}$ Fourier coefficients at the time interval $i : X_{i;0}, \dots, X_{i;\frac{l}{2}-1}$ can be computed from the previous time interval $i - 1 : X_{i-1;0}, \dots, X_{i-1;\frac{l}{2}-1}$ using:

$$\begin{pmatrix} X_{i;0} \\ X_{i;1} \\ . \\ . \\ X_{i;\frac{l}{2}-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & . & 0 \\ 0 & \upsilon^{-1} & . & . \\ . & 0 & . & . \\ . & . & . & 0 \\ 0 & . & . & \upsilon^{-(\frac{l}{2}-1)} \end{pmatrix} \cdot \begin{pmatrix} X_{i-1;0} + x_i - x_{i-w} \\ X_{i-1;1} + x_i - x_{i-w} \\ . \\ . \\ X_{i-1;\frac{l}{2}-1} + x_i - x_{i-w} \end{pmatrix} \tag{3.9}$$

with the definition of $\upsilon^k = e^{-j2\pi k/n}$ and imaginary number $j = \sqrt{-1}$. In this representation each Fourier coefficient at time interval i can be independently computed from time $i - 1$ using only $O(1)$ complex multiplications and summations:

$$X_{i;f} = \upsilon^{-f}(X_{i-1;f} + (x_i - x_{i-w})) \tag{3.10}$$



breakpoints

a \ β_i	$\beta_2(a)$	$\beta_3(a)$	$\beta_4(a)$	$\beta_5(a)$
0	-∞	-∞	-∞	-∞
1	-4,09	-4,12	-3,31	-3,62
2	-1,29	-0,49	-1,5	-0,8
3	1,55	1,54	0,57	1,36
4	3,68	3,15	2,28	2,65
5	4,73	5,98	3,43	4,24
6	+∞	+∞	+∞	+∞

Figure 3.6 – SFA quantization intervals and corresponding breakpoints for the Koski ECG dataset, an alphabet of size $c = 6$ (symbols A-F) and a symbolic word length $l = 4$. The first two columns are omitted as the first Fourier coefficient X_0 is 0 for z-normalized time series.

Computational Complexity: By the use of the MFT, the computational complexity to compute SFA words of length l is reduced to $O(l)$ for all but the first window, which has a complexity of $O(w \log w)$ when using the DFT. The length l is upper bound by 32 and can be considered constant. Thus, the computational complexity for all $O(n)$ windows is reduced from $O(n \cdot w \log w)$ to:

$$O(nl + w \log w) = O(nl + w \log w), \text{ for } l \leq 32 \quad (3.11)$$
$$= O(n + w \log w) \quad (3.12)$$

When transforming N time series of length n with window length w the computational complexity is:

$$T(FT_{subsequence}) \, \epsilon \, O(N \cdot n + w \log w) \quad (3.13)$$

3.3.4 Quantization

A time series is transformed to its SFA word by applying the Fourier transform and a simple lookup using the precomputed MCB quantization intervals.

Definition 8. *Quantization Intervals (bins)*: The a-th quantization interval QI is defined by its upper $\beta(a)$ and lower breakpoint $\beta(a-1)$ and labeled by the a-th $symbol_a$ of the alphabet Σ:

$$QI(a) = [\beta(a-1), \beta(a)) \triangleq symbol_a \, \epsilon \, \Sigma \quad (3.14)$$

As we are using different quantization intervals for each value of the Fourier transform, an index j is used to indicate the quantization intervals corresponding to the j-th numeric value of the Fourier transform $DFT(T) = t'_0, \ldots, t'_j, \ldots, t'_{l-1}$:

$$QI_j(a) = [\beta_j(a-1), \beta_j(a)) \triangleq symbol_a, \quad j \,\epsilon\, [0 \ldots l)\,,\, a \,\epsilon\, [1 \ldots c)\,, |\Sigma| = c \tag{3.15}$$

We omit the details to obtain the intervals for now and define the SFA lookup using precomputed MCB quantization intervals first. Let us assume for now that MCB quantization training returns l sets of c quantization intervals for an SFA word length l. These intervals are given by:

$$MCB = \begin{pmatrix} -\infty & -\infty & \ldots & -\infty & -\infty \\ \beta_0(1) & \beta_1(1) & \ldots & \beta_{l-2}(1) & \beta_{l-1}(1) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \beta_0(c-1) & \beta_1(c-1) & \ldots & \beta_{l-2}(c-1) & \beta_{l-1}(c-1) \\ +\infty & +\infty & \ldots & +\infty & +\infty \end{pmatrix} \tag{3.16}$$

The intervals map to the first $\frac{l}{2}$ Fourier coefficients ($\frac{l}{2}$ real and $\frac{l}{2}$ imaginary values). For z-normalized time series the values in the first two columns are 0 and can be omitted. Figure 3.6 shows 4 sets with 6 quantization intervals each, and the corresponding breakpoints for the z-normalized Koski ECG dataset [42].

Definition 9. *SFA Word*: The symbolic representation $SFA(T) = s_0, \ldots, s_{l-1}$ of a time series T with approximation $DFT(T) = real_0, imag_0, \ldots, real_{\frac{l}{2}-1}, imag_{\frac{l}{2}-1} = t'_0, \ldots, t'_{l-1}$ is a mapping $SFA : \mathbb{R}^l \rightarrow \Sigma^l$ of a real value to a symbol over the alphabet $\Sigma = \{symbol_1, \ldots, symbol_c\}$ of size c. Specifically, the j-th numeric value $t'_j \epsilon DFT(T)$ is mapped to the a-th symbol of the alphabet Σ, if it falls into the corresponding interval $QI_j(\mathrm{c}) = [\beta_j(a-1), \beta_j(a))$:

$$s_j = \Big\{ symbol_a, \quad if \;\; \Big(\beta_j(a-1) \leq t'_j < \beta_j(a)\Big) \tag{3.17}$$

, for $j \,\epsilon\, [0 \ldots l)$.

Figure 3.3 bottom right illustrates this mapping and the MCB quantization intervals. The resulting SFA word is $DAAC$ for $DFT(T) = (1.89, -4.73, -4.89, 0.56)$.

3.3.5 Quantization Training: Multiple Coefficient Binning

The aim of *multiple coefficient binning* (MCB) is to minimize the loss of information introduced by quantization, since a better description of the original signals improves pruning during query execution. This is done by applying l quantizations, given an SFA word length l, where the j-th set of quantization intervals is applied to the j-th numeric value of the DFT approximations. These map to either the real or the imaginary part of the Fourier transform.

The MCB quantization intervals are computed from a set of representative time series using a matrix of sorted Fourier coefficients.

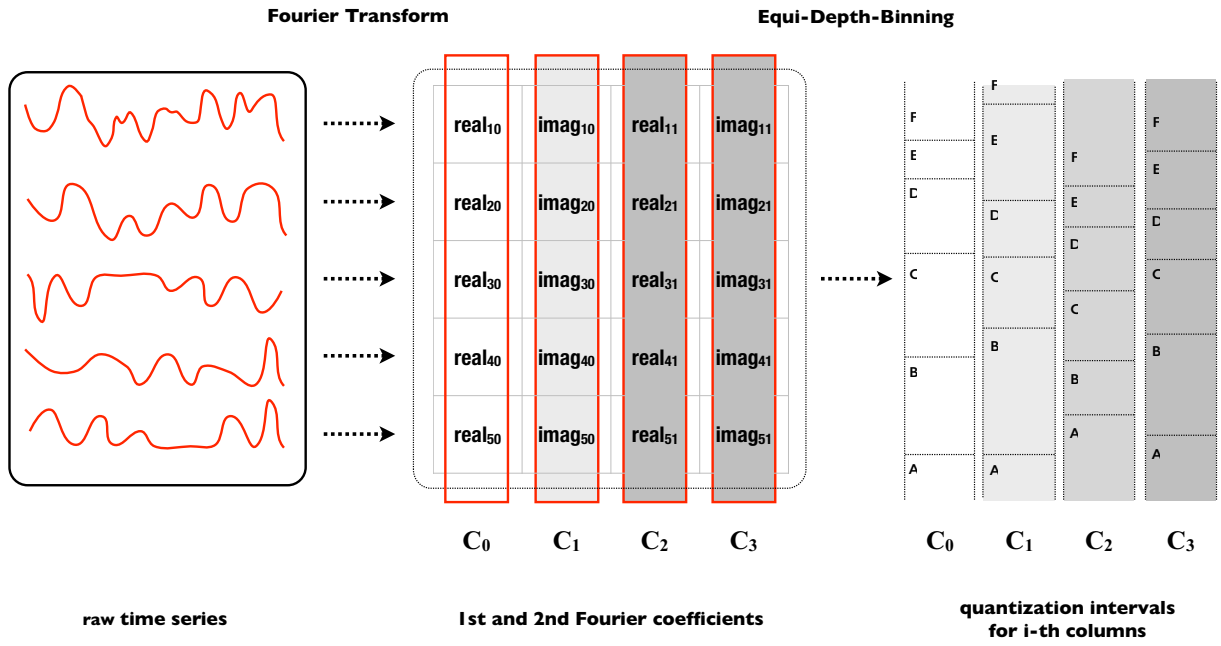


Figure 3.7 – MCB applies binning to the distribution of all i-th columns.

Definition 10. *MCB matrix*: A matrix $A = (a_{ij})_{i=1..N;j=0..l-1}$ is built from the Fourier transformations of the N train samples $T_i, i\epsilon\,[1\ldots N]$ using only the first $\frac{l}{2}$ Fourier coefficients - equal to an SFA word of length l with $\frac{l}{2}$ real and $\frac{l}{2}$ imaginary values. The i-th row of matrix A corresponds to the Fourier transform of the i-th sample T_i (compare Figure 3.7):

$$A = \begin{pmatrix} DFT(T_1) \\ . \\ DFT(T_i) \\ . \\ DFT(T_N) \end{pmatrix} \tag{3.18}$$

$$= \begin{pmatrix} real_{1;0} & imag_{1;0} & \ldots & real_{1;\frac{l}{2}-1} & imag_{1;\frac{l}{2}-1} \\ . & . & \ldots & . & . \\ real_{i;0} & imag_{i;0} & \ldots & real_{i;\frac{l}{2}-1} & imag_{i;\frac{l}{2}-1} \\ . & . & \ldots & . & . \\ real_{N;0} & imag_{N;0} & \ldots & real_{N;\frac{l}{2}-1} & imag_{N;\frac{l}{2}-1} \end{pmatrix} \tag{3.19}$$

$$= \begin{pmatrix} C_0 & C_1 & \ldots & C_{l-2} & C_{l-1} \end{pmatrix} \tag{3.20}$$

The j-th column C corresponds to either the real or the imaginary values of all N train samples. Each column is sorted by value and then partitioned into c equi-depth bins.

Definition 11. *Multiple Coefficient Binning (MCB)*: Given the sorted columns $C_j\epsilon A$ with $j\epsilon\,[0\ldots l)$, and a finite alphabet $\Sigma = \{symbol_1, \ldots, symbol_c\}$ of size c: MCB determines

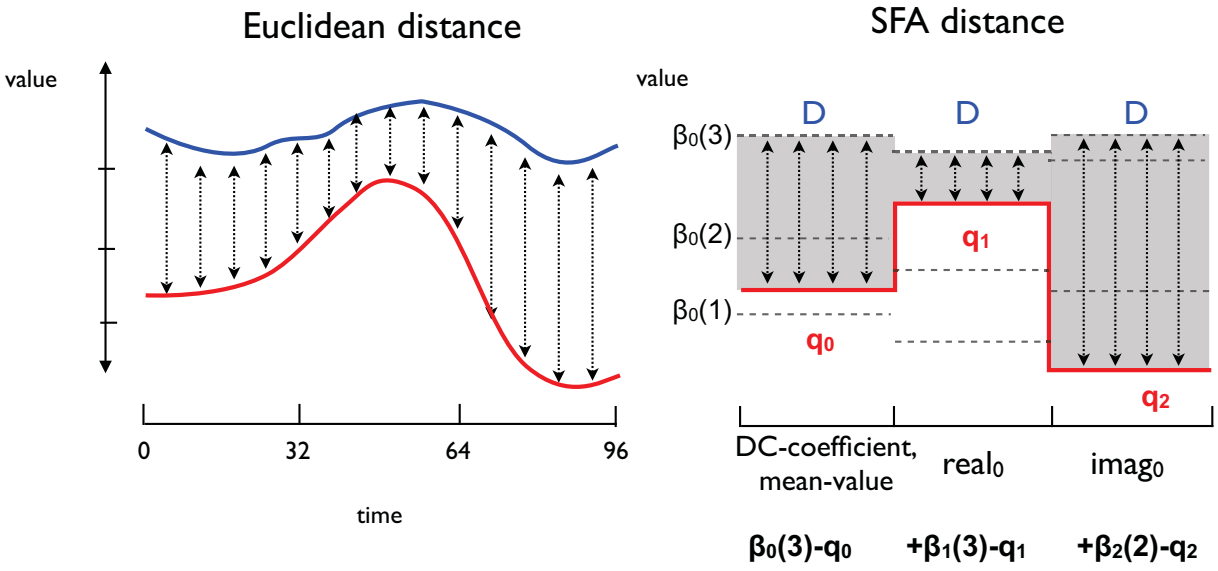


Figure 3.8 – Euclidean distance of two time series C and Q compared to the SFA distance.

$c+1$ breakpoints $\beta_j(0) < \ldots < \beta_j(c)$ for each column C_j, by applying equi-depth binning. Using an alphabet of size c and $\frac{l}{2}$ Fourier coefficients, MCB results in a total of l sets of $c+1$ intervals:

$$\beta_j(0) < \ldots < \beta_j(c), \quad for\ j \epsilon\ [0 \ldots l) \tag{3.21}$$

with the definitions $\beta_i(0) = -\infty$ and $\beta_i(c) = +\infty$. Two adjacent breakpoint define a *quantization interval* (bin):

$$QI_j(a) = [\beta_j(a-1), \beta_j(a))\,, \quad for\ j \epsilon\ [0 \ldots l)\,,\ a \epsilon\ [1 \ldots c) \tag{3.22}$$

Finally, we *label* each bin by assigning the a-th symbol of the alphabet to it:

$$QI_j(a) \triangleq symbol_a, \quad for\ j \epsilon\ [0 \ldots l)\,,\ a \epsilon\ [1 \ldots c) \tag{3.23}$$

These MCB quantization intervals have to be obtained from a set of representative train time series. For an alphabet of size c, binning results in a total of $c \cdot l$ quantization intervals.

3.3.6 Euclidean Lower Bounding Distance

Within the context of time series indexing the distance of two time series Q and C of length n is commonly measured in terms of the *Euclidean distance* (Figure 3.8 left):

$$D^2_{ED}(Q, C) = \sum_i (q_i - c_i)^2 \tag{3.24}$$

Given the DFT representations of two time series $C_{DFT} = (c'_0, \ldots, c'_{l-1})$ of C and $Q_{DFT} = (q'_0, \ldots, q'_{l-1})$ of Q the *DFT lower bounding distance* to the Euclidean distance is defined as [61] (for $l \ll n$):

$$D^2_{DFT}(C_{DFT}, Q_{DFT}) = (c'_0 - q'_0)^2 + 2\sum_{i=1}^{l-1}(c'_i - q'_i)^2 \tag{3.25}$$

This equation is a result of the complex conjugate property of the Fourier transform (Equation 3.6). Note that the first DFT coefficients c'_0 and q'_0 (DC coefficients) can be discarded to obtain z-normalization.

The *SFA Euclidean lower bounding distance* between a DFT representation $Q_{DFT} = (q'_0, \ldots, q'_{l-1})$ and an SFA representation $C_{SFA} = (c''_0, \ldots, c''_{l-1})$ is calculated by exchanging the pairwise difference of the numerical values in Equation 3.25 by a $dist_i$ function, which measures the distance between the i-th symbol and the i-th numerical value:

$$D^2_{SFA}(C_{SFA}, Q_{DFT}) \equiv dist_i(c''_i, q'_i)^2 + 2\sum_{i=1}^{l-1} dist_i(c''_i, q'_i)^2 \tag{3.26}$$

The distance $dist_i$ between a numerical value q'_i and a symbol c''_i, represented by its lower and upper breakpoints $QI_i = [\beta_i(a-1), \beta_i(a))$, is defined as the distance to the lower breakpoint if q'_i is smaller or the upper quantization breakpoint if q'_i is larger:

$$dist_i(c''_i, q'_i) \equiv \begin{cases} 0, & \text{if } q'_i \,\epsilon\, QI_i(a) \\ \beta_i(a-1) - q'_i, & \text{if } q'_i < \beta_i(a-1) \\ q'_i - \beta_i(a), & \text{if } q'_i > \beta_i(a) \end{cases} \tag{3.27}$$

Figure 3.8 (right) illustrates the MCB $dist_i$ definition.

Proof of Correctness: To prove the correctness of the SFA Euclidean lower bounding distance in Eq. 3.26 and Eq. 3.27, we have to show that it lower bounds the Euclidean distance.

Claim 1. *The distance measure D^2_{SFA} for two time series $Q_{DFT} = (q'_0, \ldots, q'_{l-1})$ and $C_{DFT} = (c'_0, \ldots, c'_{l-1})$, $C_{SFA} = (c''_0, \ldots, c''_{l-1})$ holds the Euclidean lower bounding lemma:*

$$D^2_{SFA}(C_{SFA}, Q_{DFT}) \;\leqslant\; D^2_{ED}(Q, C)$$

Proof. The distance $dist_i(c''_i, q'_i)$ with MCB breakpoints $\beta_i(a-1) \leq c''_i < \beta_i(a)$ is always smaller than the distance of two DFT-transformed coefficients q'_i and c'_i. There are two cases. First, we show $q'_i < \beta_i(a-1)$:

$$\begin{aligned} dist^2_i(c''_i, q'_i) &= (\beta_i(a-1) - q'_i)^2 \\ &\quad \textit{with } (\beta_i(a-1) \leq c'_i < \beta_i(a)) \\ &\leq (c'_i - q'_i)^2 \end{aligned}$$

In analogy the same is true for the second case $q'_i > \beta_i(a)$:

$$\begin{aligned} dist_i^2(c''_i, q'_i) &= (q'_i - \beta_i(a))^2 \\ &\quad with\ (\beta_i(a-1) \leq c'_i < \beta_i(a)) \\ &< (q'_i - c'_i)^2 \end{aligned}$$

This yields:

$$\begin{aligned} D_{ED}^2(C, Q) &\geqslant D_{DFT}^2(C_{DFT}, Q_{DFT}) && (3.28) \\ &= (c'_0 - q'_0)^2 + 2\sum_{i=1}^{l-1}(c'_i - q'_i)^2 && (3.29) \\ &\geqslant dist_i^2(c''_i, q'_i)^2 + 2\sum_{i=1}^{l-1} dist_i^2(c''_i, q'_i)^2 && (3.30) \\ &= D_{SFA}^2(C_{SFA}, Q_{DFT}) && (3.31) \end{aligned}$$

where Eq. 3.28 has been proven in [61] and Eq. 3.29 has been defined in [61]. □

3.3.7 Computational Complexity

		DFT/MFT	SFA	PAA	SAX
Whole	Pre-processing	-	$O(N(\log N + n \log n))$	-	$O(1)$
Whole	Transformation	$O(n \log n)$	$O(n \log n)$	$O(n)$	$O(n)$
Subseq.	Pre-processing	-	$O(N(\log N + n))$	-	$O(1)$
Subseq.	Transformation	$O(n + w \log w)$	$O(n + w \log w)$	$O(n)$	$O(n)$

Table 3.1 – Computational complexity of dimensionality reductions for N: number of train time series and n: time series length. The complexity depends on the use case: subsequence matching or whole matching.

Pre-processing: When Fourier transforming N time series of length n using the DFT with complexity $O(n \log n)$, the total complexity is:

$$T(FT_{whole}) \, \epsilon \, O(N \cdot n \log n) \tag{3.32}$$

When Fourier transforming N time series of length n with window length w using the MFT, the total complexity is:

$$T(FT_{subsequence}) \, \epsilon \, O(N \cdot n + w \log w) \tag{3.33}$$

The calculation of the MCB quantization intervals requires building a matrix containing all N Fourier transformed time series of length l. In this matrix, the columns are sorted and partitioned into equi-depth bins. Sorting l columns with N values has a computational complexity of $O(l \cdot N \log N)$ when using Quicksort or Mergesort. Equi-depth binning, applied to sorted columns, requires a single scan of each column with a total computational complexity of $O(l \cdot N)$. The length l is upper bound by 32 and c is fixed to 256. That results in a total computational complexity of:

$$\begin{aligned} T(MCB) &\epsilon T(Sort) + T(EquiDepth) + T(FT) && (3.34)\\ &\epsilon O(l \cdot N \log N) + O(l \cdot N) + T(FT) && (3.35) \end{aligned}$$

or for whole matching using the DFT:

$$\begin{aligned} T(MCB_{whole}) &\epsilon O(l \cdot N \log N + l \cdot N + N \cdot n \log n) && (3.36)\\ &= O(N \cdot (l \cdot \log N + l + n \log n)), \textit{ for } l \leq 32 && (3.37)\\ &= O(N \cdot (\log N + n \log n)) && (3.38) \end{aligned}$$

and for subsequence matching using the MFT:

$$\begin{aligned} T(MCB_{subsequence}) &\epsilon O(l \cdot N \log N + l \cdot N + N \cdot n + w \log w) && (3.39)\\ &= O(N \cdot (l \cdot \log N + l + n) + w \log w), \textit{ for } l \leq 32 && (3.40)\\ &= O(N \cdot (\log N + n) + w \log w) && (3.41) \end{aligned}$$

This means that the pre-processing phase to obtain the MCB quantization bins is dominated by the Fourier transforms of the N signals and the sorting step prior to the equi-depth binning.

SFA transformation: The Fourier transform of a *single* time series takes $O(n \log n)$ when using the DFT and $O(n)$ when using windowing with MFT. An SFA word of length l requires l lookups in the MCB breakpoints for the c intervals. Assuming we use binary search for those lookups this results in $O(l \cdot \log c)$ operations. This leads to a total complexity of:

$$\begin{aligned} T(SFA) &\epsilon T(FT) + T(Lookup) && (3.42)\\ &\epsilon T(FT) + O(l \cdot \log c) && (3.43) \end{aligned}$$

or for whole matching using the DFT:

$$\begin{aligned} T(SFA_{whole}) &\epsilon O(n \log n + l \cdot \log c), \textit{ for } l \leq 32, c \leq 256 && (3.44)\\ &= O(n \log n) && (3.45) \end{aligned}$$

and for subsequence matching using the MFT:

$$\begin{aligned} T(SFA_{subsequence}) &\epsilon O(n + l \cdot \log c + w \log w), \textit{ for } l \leq 32, c \leq 256 && (3.46)\\ &= O(n + w \log w) && (3.47) \end{aligned}$$

This means that the SFA transformation is dominated by the Fourier transform and there is a negligible impact by the SFA word length l, the window length, or the alphabet size c.

Comparison: Table 3.1 illustrates the computational complexities of SAX, the Fourier transform (DFT, MFT), and SFA. For subsequence matching SFA is very competitive with regards to transformation times, but slower than SAX for whole matching. Most time series problems are subsequence matching problems.

3.3.8 Space Complexity

Technique	Numerical	SFA	SAX/iSAX
Time	$O(Nl64)$	$O(Nl\log_2 c)$	$O(Nl\log_2 c)$

Table 3.2 – Asymptotic space complexity in bits of dimensionality reductions for N time series of length l, and an alphabet of size c.

The memory footprint of SFA depends on the SFA word length l and the alphabet size c. An SFA word of length l requires $l\log_2 c$ bits. Thus, an alphabet of size $c = 256$ can be encoded in $\log_2 256$ bits or 1 byte. Given N time series this results in $Nl\log_2 c$ bits. Additionally SFA requires an overhead of $l(c-1)\cdot 8$ bytes for storing the l lookup tables containing the $c-1$ real-valued breakpoints with 8 bytes (double) each. Since these lookup tables are stored once, the overhead in terms of memory is negligible for large N, resulting in a total complexity of

$$O(Nl\log_2 c)$$

bits. E.g., given N SFA words of length l and alphabet of size $c = 256$, SFA requires $O(Nl)$ bytes.

Comparison to SAX/iSAX: SFA has the same asymptotic space complexity as iSAX: for an alphabet of size c, iSAX/SAX requires $l\log_2 c$ bits for each SAX word. Additionally SAX requires a small overhead of $(c-1)\cdot 8$ bytes for the lookup table containing the $c-1$ real-valued breakpoints with 8 bytes (double) each. Following the same reasoning as for SFA, this overhead is negligible for large N, resulting in a total complexity of

$$O(Nl\log_2 c)$$

bits. E.g., given N SAX words of length l and alphabet of size $c = 256$, SAX requires $O(Nl)$ bytes.

Comparison to numeric dimensionality reduction techniques: In case of numeric dimensionality reductions each value of the approximation is typically stored in an 8 byte double, resulting in a total of $l \cdot 8$ bytes for each approximation. A symbolic representation with $c = 256$ symbols requires l bytes, or even less using less symbols. Symbolic representations allow for indexing terabyte-sized data in main memory [82, 20] due to this at least 8-fold lower memory footprint by utilizing quantization on top of approximation (compare Table 3.2).

3.4 Indexing SFA

The purpose of indexing is to partition the search space into roughly equal-sized groups containing similar entries and to provide a fast lookup for these entries. The idea of indexing SFA is simple: grouping is done based on a prefix of the first k characters of the SFA words. Increasing the prefix length k leads to smaller sized groups. As the entries are not equally distributed in the search space, the prefix length k is individually increased until all entries are equally distributed over each group.

SFA is based on the quantization of the Fourier transformation. Each Fourier coefficient individually contributes to the *global information* on the shape of a time series. By adding coefficients to a k-sized group, we can add further detail (reduce the reconstruction error) without the need to recalculate the first k Fourier coefficients or the other groups being affected. The first lower frequencies provide more information about the time series (*inherent ordering* of the Fourier coefficients). Due to this inherent ordering, we can just add the $(k+1)$-th coefficient to a k-length group for indexing (*simple split criterion*).

This means that SFA is best computed at a high SFA word length, but only a few groups exploit the full SFA word length and most use only a small prefix.

The inherent ordering and the global information are maintained by the SFA transformation and the SFA trie:

- *inherent ordering*: the k-th level of the index equals the k-th character of the SFA word.
- *global information*: splitting a leaf node at the k-th level is performed by adding the $(k+1)$-th character, thus increasing the depth of the leaf node by one.

In essence, the SFA trie is a *prefix tree* (called *trie*) built over a prefix of the SFA word. A classical prefix tree is defined as follows:

- *Edges*: Each edge is labeled by a symbol of the alphabet $\Sigma = \{symbol_1, \ldots, symbol_c\}$.
- *Internal Node*: Each node is labeled by a prefix $s_0 \ldots s_k$ with $s_i \, \epsilon \, \Sigma$ that is defined by concatenating the labels on the edges from the root node to that node. An internal node has at most $|\Sigma|$ children. Given an internal node with the prefix $s_0 \ldots s_k$, its children have the prefixes $s_0 \ldots s_k a$, with $s \, \epsilon \, \Sigma$ being the label on the incoming edge.

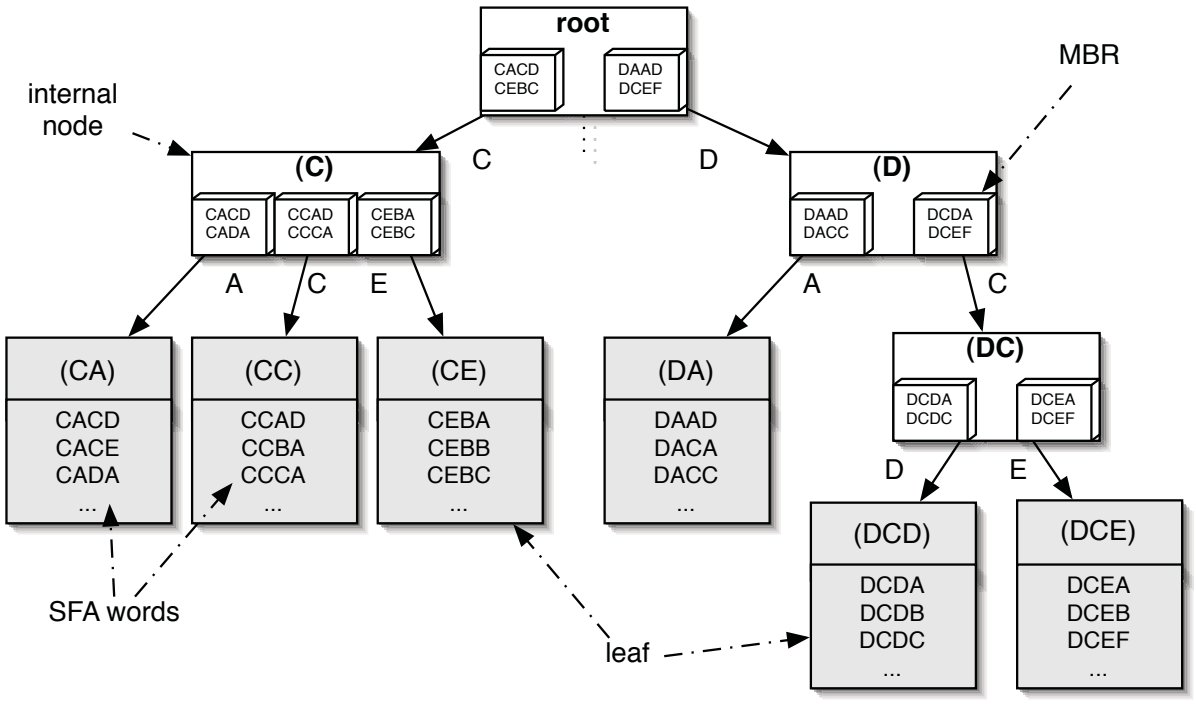


Figure 3.9 – SFA trie: for simplicity the MBRs show SFA symbols instead of Fourier coefficients.

- *Leaf Node*: Leaf nodes contain words. The leaf node labeled $s_0 \dots s_k$ contains all words which share the prefix $s_0 \dots s_k$. For any key, built over the alphabet Σ, there is at most one leaf node, which is labeled by a prefix of that key (*overlap free*).
- *Root Node*: The root node is an internal node, which is labeled by an empty prefix.

In the SFA trie, each node represents the time series sharing the same prefix of an SFA word. The *SFA trie* has the following modifications/properties:

1. *Controlled Fanout*: Each node has at most $|\Sigma|$ children.
2. *Overlap-free*: The space is partitioned using the SFA alphabet Σ. This guarantees that the trie nodes are overlap-free, which is an important property of an index [10]. *Overlap-free* means that for insertion, querying or deletion exactly one branch of the tree has to be iterated.
3. *Threshold*: A threshold th is used to control the maximal number of time series for each leaf node until it is split. This guarantees that the paths within the trie do not get too sparse.
4. *Split*: A split is performed after the $(th+1)$-th element is inserted into a leaf node. Splitting a leaf node with prefix length k is performed by relabeling the leaf node as

an internal node and simply reinserting the SFA words into this node using a prefix length $k+1$.

5. *Controlled Depth*: The depth of the SFA trie is limited by the SFA word length l.

6. *Pointers*: Each leaf node contains a pointer to the original time series stored on the disk.

7. *MBR*: Each internal node V with prefix $s_0 \dots s_k$ contains a minimum bounding rectangle (MBR), which is an l-dimensional array of tuples:

$$MBR = [(min_0, max_0) \dots (min_{l-1}, max_{l-1})] \tag{3.48}$$

where l equals the SFA word length. It contains the real-valued upper and lower bounds for each dimension of the DFT transformed time series contained in all children of V.
Given the DFT representations $DS_{DFT} = \{T'_1, \dots, T'_n\}$, with $T'_i = (t'_0, \dots, t'_{l-1})$, of the SFA prefix $s_0 \dots s_k$: The a-th tuple $(\min_a, \max_a)$ of an MBR is defined by the minimum and maximum over the a-th dimension:

(a) $\min_a = \min\{t'_a \,|\, T'_j = (t'_0, \dots, t'_a, \dots, t'_{l-1}) \,\epsilon\, DS_{DFT}\}$ and
(b) $\max_a = \max\{t'_a \,|\, T'_j = (t'_0, \dots, t'_a, \dots, t'_{l-1}) \,\epsilon\, DS_{DFT}\}$.

The SFA trie is a prefix tree (Figure 3.9). The trie makes use of MBRs, a threshold and a simple split criterion in combination with the Fourier frequency domain. The SFA trie is not height balanced but has a maximal fanout equal to the alphabet size.

The novelty of the trie is that it adapts to dense leaf nodes by increasing the length of the prefix of the SFA words, which equates to improving the quality of the representations without altering all other neighboring leaf nodes. This increases the accuracy of the similarity search, i.e., less leaf nodes have to be inspected to answer a similarity query.

The SFA trie has three parameters:

- The *threshold th*: It controls the maximal number of time series for each leaf node until it is split. It is typically set to (multiples of) the block size.

- The *SFA alphabet size* $c = |\Sigma|$: It controls the fanout of the SFA trie. We empirically found $c = 8$ to be a good value.

- The *SFA word length* l: It controls the maximal height of the SFA trie. We test the effects of this parameter in our experiments (Chapter 3.5).

Algorithm 3.1 SFA trie insertion.

```
1  public void insert(TimeSeries T, short[] T_SFA, double[] T_DFT, int k, Node node)
2    Node child = node.getChild(T_SFA[k])
3    // insert into existing node
4    if (child != null)
5      child.adaptMBR(T_DFT)
6      if (child.isInternal())
7        insert(T, T_SFA, T_DFT, k+1, child)
8      else if (child.isLeaf())
9        child.add(T)
10       if (child.split())
11         child.setInternal()
12         for (ts, tsSFA, tsDFT) in child
13           insert(ts, tsSFA, tsDFT, k+1, child)
14   // add new node
15   else
16     Node newchild = new Leaf
17     node.add(T_SFA[k], newchild)
18     newchild.add(T)
```

3.4.1 Insertion

Algorithm 3.1 illustrates the *insert*-method of the SFA trie. Given a time series T, its SFA representation T_{SFA} and the DFT representation T_{DFT}, we have to find the one leaf node, whose label is a prefix of T_{SFA}. Given a node at depth k, we retrieve the one child node, which has the same prefix (line 2). Next, we adapt the MBR of that child using the current DFT representation (line 5).

If that child is an internal node (line 6), we recursively insert the time series into that node and increase the length of the prefix by one. If that child is a leaf node (line 8), we add the time series to the leaf and check if the insertion caused the leaf to split (line 10). For splitting, we flag the leaf node as an internal node, and reinsert all time series C into that node (lines 12–13), causing the length of the prefix to grow by one.

If the child node does not exist, we create a new child and add the time series T (lines 15–18).

3.4.2 Computational Complexity

The number of operations for the *insert*-method is bound by the depth of the SFA trie which is limited by the maximal SFA word length l. Thus, descending the SFA trie to find the corresponding leaf node requires at most $O(l)$ operations, assuming we use hashing for storing the (label,edge)-pairs with a lookup complexity of $O(1)$ in line 2:

$$T(getChild) \, \epsilon \, O(l) \tag{3.49}$$

Due to the need to calculate the minimal and maximal value for each of l dimensions, adjusting the MBRs (line 5) requires $O(l)$ operations for each level of the trie:

$$T(adaptMBR) \quad \epsilon \quad O(l) \tag{3.50}$$

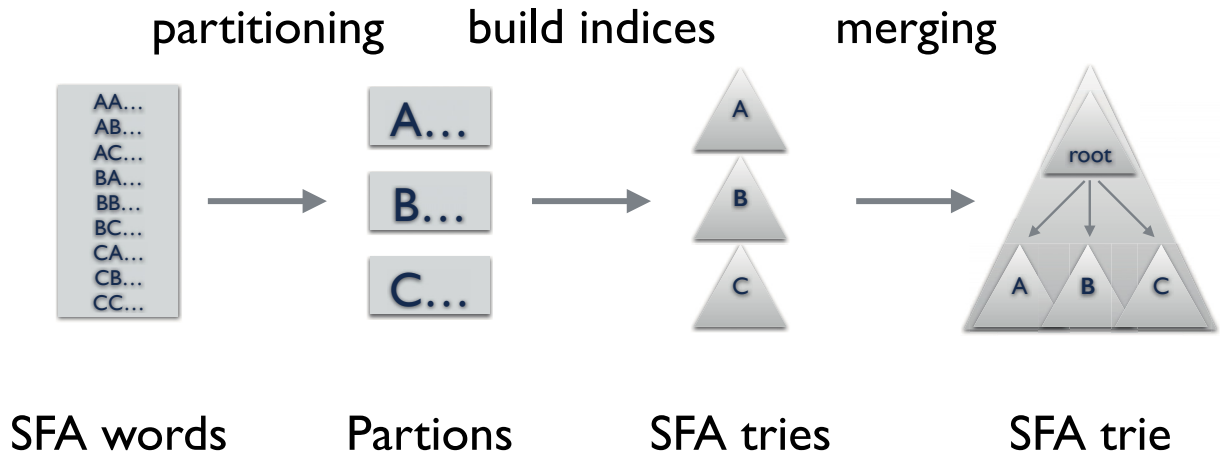


Figure 3.10 – Bulk loading into an SFA trie.

A node split is applied locally (lines 10-13) and a total of $th + 1$ time series have to be reinserted causing the corresponding MBRs to adjust. This leads to a total of $O(T(adaptMBR) \cdot th)$ operations in case of a split:

$$T(split) \quad \epsilon \quad O(T(adaptMBR) \cdot th) \tag{3.51}$$
$$= \quad O(l \cdot th) \tag{3.52}$$

However, this *split*-operation happens only every $th + 1$ *insert*-operations with complexity $O(l \cdot th)$, whereas all remaining th *insert*-operations have a complexity of $O(l)$. This leads to an amortized computational complexity of:

$$T(splitAmortized) \quad = \quad \frac{th}{th} \cdot T(noSplit) + \frac{1}{th} \cdot T(split) \tag{3.53}$$
$$\epsilon \quad O(\frac{th}{th} \cdot l + \frac{1}{th} \cdot l \cdot th) \tag{3.54}$$
$$= \quad O(l) \tag{3.55}$$

This results in an amortized computational complexity for the SFA trie insertion and SFA transformation of a time series of length n with SFA words of length l of:

$$T(insert) \quad = \quad T(SFA) + T(getChild) \cdot T(adaptMBR) + T(splitAmortized) \tag{3.56}$$
$$\epsilon \quad T(SFA) + O(l^2 + l) \tag{3.57}$$
$$= \quad T(SFA) + O(l^2) \tag{3.58}$$

The *insert*-method is therefore dominated by the *adaptMBR*-method in line 5.

3.4.3 Bulk Insertion

Bulk insertion of a large amount of time series can be a challenging task, if the main memory size is not sufficient to hold all time series. In that case, the data has to be

Algorithm 3.2 SFA trie Bulk Insertion.

```
1  public SFATrie bulkInsert(TimeSeries[] DS, char[] partitions)
2    // obtain SFA words and sort
3    SFAWords[] sfa_words = []
4    for TimeSeries ts in DS // in parallel
5      sfa_words.append(SFA(ts))
6    // sort into buckets in parallel
7    TimeSeries[][] buckets = bucket_sort(partitions, sfa_words)
8    // partition data and build separate indices
9    SFATrie[] indices = []
10   for TimeSeries[] bucket in buckets // in parallel
11     SFATrie index = new SFATrie()
12     for TimeSeries ts in bucket
13       index.insert(ts)
14     writeToDisk(index)
15     indices.append(index)
16   // merge SFA tries
17   SFATrie index = new SFATrie()
18   for each SFATrie partitionIndex in indices
19     readInternalNodesFromDisk(partitionIndex)
20     merge(index.root, partitionIndex.root, 0)
21   return index
```

written to disk while index construction, and each new time series insertion will lead to a random disk access.

However, bulk insertion of large amounts of time series into an SFA trie is simple if we make use of an important observation: the structure of the SFA trie is independent from the order of insertion of the time series: there are no rebalancing operations and, due to the inherent ordering, the simple split criterion is independent of the shape of the time series. This allows for *parallel*, *in-memory* construction of the index based on partitions of the time series (Figure 3.10). The index for each partition is written to disk after construction. The final SFA trie is built from these partitions. In this way we efficiently reduce the number of random disk operations needed.

Build partitions of the data: Algorithm 3.2 shows the *bulk-insert*-method for a set of time series DS. Input parameters are the time series dataset DS and the partitions to use, i.e., [A,B,C] in (Figure 3.10). First, all time series are transformed to their corresponding SFA words (lines 4–5). Next, all SFA words are sorted into buckets according to their SFA word prefix (line 7). These buckets are used to build separate SFA tries (lines 10–15). The size of the partitions should roughly equal the amount of available main memory. It can be adapted by using longer prefixes. These indices are stored to disk.

Finally, the nodes of the indices are merged pairwise (lines 18–20). When loading the index, it is sufficient to read the internal nodes (line 19). The leaf nodes are read from disk on demand in Algorithm 3.3.

Parallel Execution (Algorithm 3.2): The code sections of (a) transforming each time series (lines 4ff), (b) sorting into buckets (lines 7ff), and (c) building separate indices

Algorithm 3.3 SFA trie merge.

```
1  public void merge(Node currentNode, Node toInsert, int k)
2    currentNode.adaptMBR(toInsert)
3    // no child with the same prefix, append the node
4    if (currentNode == null)
5      currentNode.addChild(toInsert)
6    // I) append to an internal node
7    else if (currentNode.isInternal())
8      Node childNode = currentNode.getChild(toInsert.sfa[k]);
9      // Ia) append an internal node:
10     // recursively merge child nodes
11     if (toInsert.isInternal())
12       for each Node child in toInsert
13         merge(childNode, child, k+1)
14     // Ib) append a leaf node: add all time series
15     else if (toInsert.isLeaf())
16       for (ts, tsSFA, tsDFT) in toInsert
17         insert(ts, tsSFA, tsDFT, k, childNode)
18   // II) append to a leaf node
19   else if (currentNode.isLeaf())
20     // IIa) append an internal node:
21     // add all children and reinsert time series
22     if (toInsert.isInternal())
23       currentNode.setInternalNode()
24       currentNode.addChild(toInsert)
25       for (ts, tsSFA, tsDFT) in currentNode
26         insert(ts, tsSFA, tsDFT, k, currentNode)
27     // IIb) append a leaf node: add all time series
28     else if (toInsert.isLeaf())
29       for (ts, tsSFA, tsDFT) in toInsert
30         insert(ts, tsSFA, tsDFT, k-1, currentNode.parentNode)
```

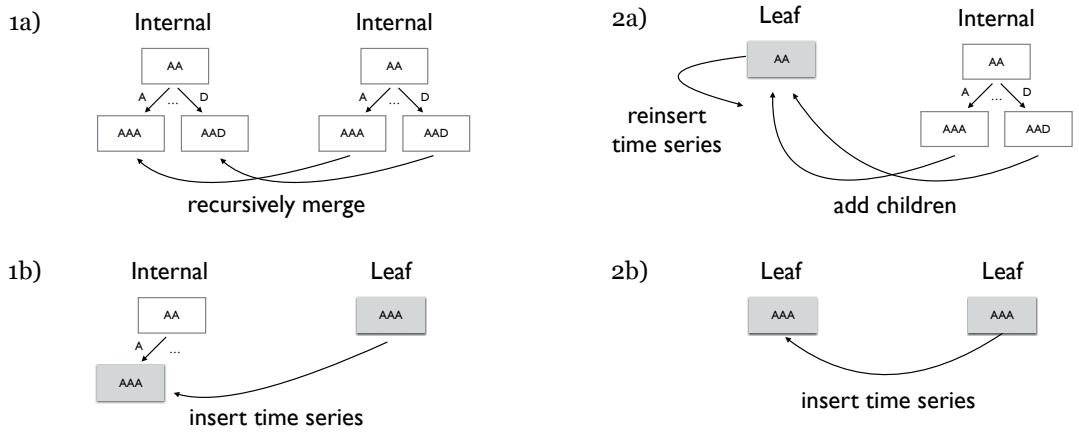


Figure 3.11 – Four cases for merging two SFA trie nodes.

(lines 10ff) are embarrassingly parallel: These operations are executed simultaneously and each one runs in a separate process. The number of simultaneous processes depends solely on the number of partitions chosen. There are three independent partitions for [A,B,C], nine partitions for [AA,AB,AC,BA,BB,BC,CA,CB,CC], or c^l for alphabet size c and prefix length l in general.

Merge the partitions of the data: Merging two SFA tries is implemented using Algorithm 3.3. Input parameters are the two nodes and the current depth for merging the two nodes. The complexity and number of disk accesses of the *merge*-method depend to a large extend on the partitions of the index. The partitions are best chosen so that each one starts with a separate prefix: i.e., [A,B,C] (compare Figure 3.10). When merging SFA tries with disjoint prefixes, the internal nodes can be appended to the root node without accessing any leaf nodes or reading time series from disk (lines 4–5).

If the prefixes of the nodes overlap, there are four cases (Figure 3.11):

1. Merge an *internal node* with:
   (a) An *internal node*: Merging two internal nodes with the same prefix is performed by recursively merging the children of the nodes (lines 11–13).
   (b) A *leaf node*: Merging an internal node with a leaf node is performed by reinserting all time series from the leaf node into the internal node. This requires to read the time series from disk once (lines 15–17).
2. Merge a *leaf node* with:
   (a) An *internal node*: Merging a leaf node with an internal node is done, by transforming the node to an internal node, adding all new child nodes and reinserting all time series into the internal node (lines 22–26).

(b) A *leaf node*: Merging two leaf nodes is done by adding all time series to one node (lines 28–30). This requires reading the time series from disk. This operation might cause the leaf to split, thus we recursively call *insert* for each time series.

3.4.4 Euclidean Lower Bounding Distance

During query processing, the distance of a query to an MBR has to be calculated. This distance has to lower bound the Euclidean distance for the GEMINI-framework to be applicable. An MBR consists of an array of l tuples with real-valued upper and lower bounds (see Equation 3.48):

$$MBR = [(min_0, max_0), \ldots, (min_{l-1}, max_{l-1})] \tag{3.59}$$

MBR distance: The distance between a DFT representation $Q_{DFT} = (q'_0, \ldots, q'_{l-1})$ and an MBR is defined in analogy to the SFA distance (Equation 3.26):

$$mindist^2(MBR, Q_{DFT}) = dist_{mbr}((min_0, max_0), q'_0)^2 + 2\sum_{i=1}^{l-1} dist_{mbr}((min_i, max_i), q'_i)^2 \tag{3.60}$$

The distance between a numerical value q'_i and a tuple (min_i, max_i) is defined as the distance to the lower bound, if q'_i is smaller, and the distance to the upper bound, if q'_i is larger. We modify Equation. 3.27 by replacing breakpoints β_i by the tuple (min_i, max_i):

$$dist_{mbr}((min_i, max_i), q'_i) \equiv \begin{cases} 0 & , \text{if } min_i \leqslant q'_i \leqslant max_i \\ min_i - q'_i & , \text{if } q'_i < min_i \\ q'_i - max_i & , \text{if } q'_i > max_i \end{cases} \tag{3.61}$$

The MBR distance (Equation 3.60) is a lower bounding distance to the DFT distance (Equation 3.26), and as such allows for the GEMINI framework (Chapter 2.4) to be applied for exact similarity search:

$$mindist(MBR, Q_{DFT}) \leq D_{DFT}(T_{DFT}, Q_{DFT}) \leq D_{ED}(T, Q) \tag{3.62}$$

3.4.5 k-Nearest-Neighbor Exact Search

Algorithm 3.4 introduces the k-NN exact search algorithm using the SFA trie and the MBR distance. Algorithm 3.5 introduces an approximate search algorithm using the SFA trie. The approximate search is used as a subroutine in the exact search algorithm. The k-NN exact search algorithm makes use of the *mindist* Euclidean lower bounding distance measure. The results of the execution of Algorithm 3.4 are exactly the same, as if the query was executed on all time series using the Euclidean distance. Our k-NN exact search is an adaptation of the multistep 1-NN exact search algorithm as introduced in [82, 78].

Algorithm 3.4 Exact k-nearest-neighbor search using the SFA trie.

```
1  public PriorityQueue<TimeSeries> exactKNNSearch(TimeSeries Q, double[] Q_DFT, short[]
       SFA, int k)
2    PriorityQueue<Node, double> queue = []
3    Queue<TimeSeries> tsResult = []
4    PriorityQueue<TimeSeries,double> tsCandidates = approximateSearch(Q,Q_SFA,root,0)
5    queue.push(root, 0)
6    while (!queue.isEmpty()) do
7      (currentNode, MBR_distance) = queue.removeMinimum()
8      // add time series to tsResult
9      for (T, ED_distance) in tsCandidates
10       if ED_distance <= MBR_distance
11         tsCandidates.remove(T)
12         tsResult.insert(T)
13     // all k-NN found?
14     if size(tsResult) >= k
15       return tsResult
16     // internal node: use "mindist"
17     if currentNode.isInternal()
18       (T, tsDist) = tsCandidates.get(k-size(tsResult))
19       for Node node in currentNode.children do
20         double MBR_distance = mindist( node.MBR, Q_DFT ) // MBR distance
21         if MBR_distance < tsDist
22           queue.push( node, MBR_distance )
23     // leaf: use Euclidean distance
24     else if currentNode.isLeaf()
25       TimeSeries[] rawTs = read time series from disk
26       for TimeSeries T in rawTs
27         tsCandidates.push( T, D_ED(T, Q) ) // Euclidean distance
28   return tsResult
```

Algorithm 3.5 Approximate search using the SFA trie.

```
1  public PriorityQueue<TimeSeries, double> approximateSearch( TimeSeries[] Q, short[]
       Q_SFA, Node currentNode, int i)
2    Node child = node.getChild( Q_SFA[i] )
3    if (child != null)
4      // follow the branch
5      if (child.isInternal())
6        return approximateSearch(Q, Q_SFA, child, i+1)
7      // return all time series in the leaf node
8      else if (child.isLeaf())
9        TimeSeries[] rawTs = retrieve raw time series from hard disk
10       PriorityQueue<TimeSeries, double> tsCandidates = []
11       for TimeSeries ts in rawTs
12         tsCandidates.push( ts, D_ED(T, Q) ) // Euclidean distance
13       return tsCandidates
14   // there is no node with the same SFA representation
15   return []
```

Input parameters of the exact search are the query Q, the DFT representation of the query Q_{DFT}, the SFA representation of the query Q_{SFA}, and the number of nearest neighbors k. A priority queue *queue* is used for index navigation and storage of the nodes having the smallest *mindist* distance. Another priority queue *tsCandidates* is used to store all raw time series currently known. The queue *tsResult* is used to store the k-NN to our query. A time series is transferred from *tsCandidates* to *tsResult* as soon as there are no nodes with a smaller *mindist* distance available (lines 10–12).

The algorithm starts by performing the approximate search (Algorithm 3.4 line 4) using the SFA trie. The approximate search (Algorithm 3.5) exploits the fact that the nearest neighbors and the query are likely to have the same SFA word. Therefore, it searches for the one leaf node with the same SFA word as the query and adds those time series to *tsCandidates* to have an upper bound for pruning time series in the SFA trie (used in Algorithm 3.4 lines 21–22).

Algorithm 3.4 starts with the root node of the index (line 5). It pops the node having the smallest *mindist* distance from the *queue* (line 7):

- If this node is an *internal node* (line 17), we traverse and insert all child nodes into *queue*, if their *mindist* is lower than the ED distance of the best known time series in *tsCandidates* (lines 17–22). The distance of the k-th best known time series is used for pruning, which is the $(k - size(tsResult))$-th element in *tsCandidates* (line 18).
- If the node is a *leaf node* (line 24), we retrieve the raw time series from disk (line 25) and insert these and their ED distance to the query into *tsCandidates* (line 27). A time series candidate can be transferred to *tsResult*, once there are no nodes left which have a lower *mindist*-distance (lines 10–12).

The exact search terminates if *queue* is empty (line 6) or we have found all k-NN (lines 14–15).

3.5 Experiments

In our experimental evaluation we show the effectiveness of SFA, the MCB quantization, and the SFA trie in terms of (a) the pruning power, and (b) exact k-NN query performance. We compare our algorithms with numeric and symbolic dimensionality reduction techniques, and spatial access methods (SAMs). We implemented all algorithms and SAMs in JAVA. All experiments were performed using a shared memory machine running LINUX with 8 Quad Core AMD Opteron 8358 SE processors, and JAVA JDK x64 1.8. For all experiments the time series datasets were z-normalized prior to the experiments (compare Chapter 2.2).

The pruning power experiments ran on exactly the same 29 datasets as presented in the iSAX (2.0+) index publications [82, 21, 20]. The exact similarity search experiments were performed on these 29 datasets and 10 additional synthetic datasets from the UCR

Motion Capture		
Kung Fu Motions	Ann_gun_CentroidA	PostureCentroidB
Trajectory3_7_2006		
Sensor Readings		
Star Light Curve	Mallet	Converted ERP
ECG (Koski ECG)	Power Data	Tickwise
Passgraph (Pen)	Chlorine	Foetal ECG
Physiology Data	ECG (mitdbx)	ECG (chfdb)
Buoy Sensor	Fluid Dynamics	Motor Current
Respiration	Industrial Winding	CSTR
Burst	Italy power demand	Muscle Activation
Synthetic / Model Data		
Boiler Model (Steamgen)	Synthetic Lightning EMP	Lightning (forte6class)
synthetic (synthetic1 to synthetic10)		

Table 3.3 – 29 Time series datasets collected from [82] and 10 synthetic datasets from the UCR data mining archive [42].

data mining archive [42]. The datasets can be grouped into 3 categories: motion capture data, sensor readings and synthetic/model data (Table 3.3). There are 4 datasets resulting from motion capture devices attached to human subjects. 21 datasets result from sensor readings, and 13 datasets are generated from a model. In all these datasets subsequences of a fixed length are extracted from long time series (subsequence matching).

The SFA publication goes along with a website [77] which contains raw numbers and some additional figures for each dataset.

3.5.1 Pruning Power

SFA is based on two design decisions:

1. The Fourier transform as a dimensionality reduction technique and
2. The MCB quantization technique.

We measured the *pruning power* to underline the utility of these two decisions. It is the standard benchmark for the performance of 1-NN queries and free of any implementation bias. The *pruning power* P is defined as the fraction of the database that must be

Algorithm 3.6 The pruning power of a dimensionality reduction technique.

```
1  public double pruningPower(TimeSeries Q, TimeSeries[] DS)
2    Map<double, TimeSeries> distances = []
3    for TimeSeries TS in DS
4      distances.append(D_lower(Q,TS), TS)
5    // sort by distance
6    sort(distances)
7    double best_distance = Double.MAX_VALUE
8    int pruningPower = 0
9    // in ascending order of their lower bounding distance
10   for (distance, TS) in distances
11     // lower bounding distance exceeds best distance
12     if (distance > best_distance)
13       return pruningPower / size(DS)
14     // continue searching
15     pruningPower++
16     best_distance = min(best_distance, D(Q,TS))
17   return pruningPower / size(DS)
```

examined using the lower bounding distance measure before the exact 1-NN to a query can be found [40] (lower is better):

$$P = \frac{number\ of\ objects\ to\ be\ examinded\ using\ D_{lower}}{total\ number\ of\ time\ series} \tag{3.63}$$

The pruning power is determined by first sorting all time series according to their lower bounding distance D_{lower} to a query (Algorithm 3.6 lines 3–6). Next, the true distance to the query is determined using the distance measure D in ascending order of D_{lower} (line 10). When the lower bounding distance exceeds the best so far true distance $D_{lower} > D$, the nearest neighbor is found (lines 12–13). This number is the absolute minimum of time series any SAM has to examine in order to find a nearest neighbor to a query (compare Chapter 2.3). In the context of time series indexing, the Euclidean distance D_{ED} is commonly used as the true distance D.

3.5.1.1 Impact of Dimensionality Reduction Technique "DFT"

It has been claimed that there is little difference between the dimensionality reduction techniques [54, 43]. However, we made a different observation: DFT is the most accurate dimensionality reduction technique. To underline this, we compare state of the art numeric dimensionality reductions and the symbolic representation SFA and SAX/iSAX in terms of their pruning power. The results were averaged over the 29 datasets for time series lengths $n = \{256, 512, 1024\}$, an increasing number of values $l = [4, 8, 16]$ to represent one dimensionality reduction technique, and an alphabet of 256 symbols for SFA and iSAX. 10% of each dataset were extracted for queries and the remaining 90% of the dataset were used to determine the pruning power. Each query is executed using Algorithm 3.6 and the results are averaged. We compared the influence of seven dimensionality reductions on the *pruning power* using:

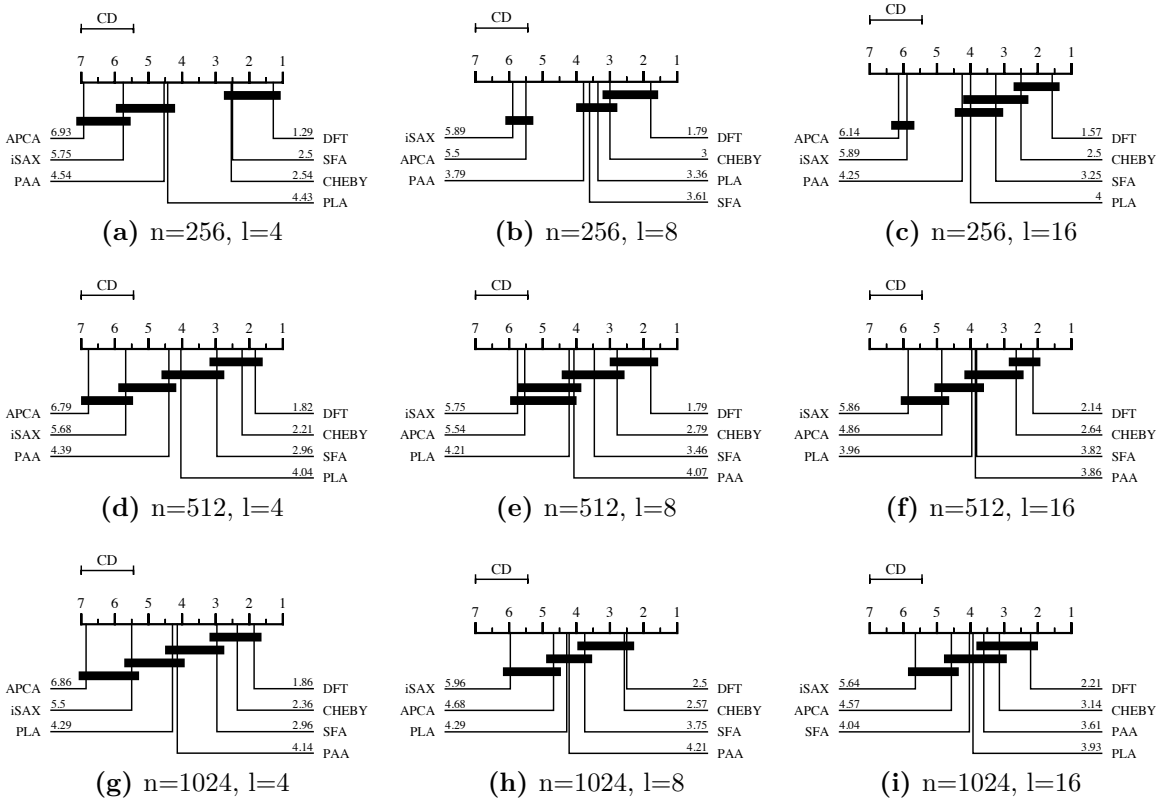


Figure 3.12 – Critical difference diagrams over the average ranks of the *pruning power* (lower is better) for state of the art dimensionality reductions on the 29 real datasets for time series lengths n=[256,512,1024], l=[4,8,16] values, and c=256 symbols.

1. Numeric dimensionality reductions: DFT, PAA, APCA, PLA, CHEBY.
2. Symbolic dimensionality reductions: *SFA, iSAX/SAX.*

Figure 3.12 shows the critical difference diagrams, as introduced in [28], over the average ranks based on the pruning power of the representations on the datasets. The classifiers with the lowest (best) ranks are to the right. The group of classifiers that are not significantly different from each other are connected by a horizontal black bar. The critical difference (CD) length is shown above each graph.

- DFT and Chebyshev are the best numerical dimensionality reduction techniques for 1-NN searches, whereas DFT has the lowest rank for all tested configurations.
- SFA makes use of DFT for approximation and as such is better than most of the other numeric representations and only slightly worse than DFT (due to the use of quantization of DFT) and competitive with CHEBY.

n	1024	1024	1024	512	512	512	512	512	512
l	16	8	4	16	8	4	16	8	4
N	28	28	28	29	29	29	29	29	29
p-value	0.0004	0.011	0.0004	0.0002	6.6e-05	6.5e-05	9.1e-05	6.2e-05	4.3e-07

Table 3.4 – One-tailed Wilcoxon signed-rank test for null-hypothesis $P_{PAA} \leq P_{DFT}$.

- When comparing the symbolic representations, SFA is two to three ranks better than iSAX.
- The representations based on the mean values PAA, iSAX, APCA perform significantly worse than the other representations.

Implications: Overall, SFA is not only better than SAX/iSAX but the results indicate that it is competitive with the best numeric dimensionality reduction techniques regarding the number of time series to be inspected. We conclude that these improvements are a result of the use of DFT approximation, as DFT has the best pruning power. The main advantage of SFA over DFT is its lower memory footprint and its symbolic representation (Chapter 3.3.8). This allows for indexing large and high dimensional datasets through the SFA trie or the use of the bag of patterns representation.

Wilcoxon Signed-Rank Test: To confirm these results, we applied a one-tailed Wilcoxon signed-rank test to check if any method has a significantly better pruning power. The null-hypothesis is that the pruning power of PAA P_{PAA} is lower (better) than the pruning power of DFT P_{DFT}:

$$H_0 : \quad P_{PAA} \leq P_{DFT} \tag{3.64}$$

A null-hypothesis can be *rejected* if the p-value is smaller than 0.05. With a p-value of at least 0.011 at configuration n=1024, l=8, the null-hypothesis H_0 can be rejected on all tested configurations (Table 3.4). The alternative hypothesis that *"DFT has a better pruning power than PAA*" is thus statistically significant.

Implications: Thus, the claim that there is little difference between dimensionality reduction techniques is statistically wrong. The alternative hypothesis that *"DFT has a better pruning power than PAA*" is statistically significant. This is a reason why we chose DFT as a dimensionality reduction technique as part of SFA, as opposed to SAX which uses PAA.

3.5.1.2 Impact of Quantization Technique "MCB"

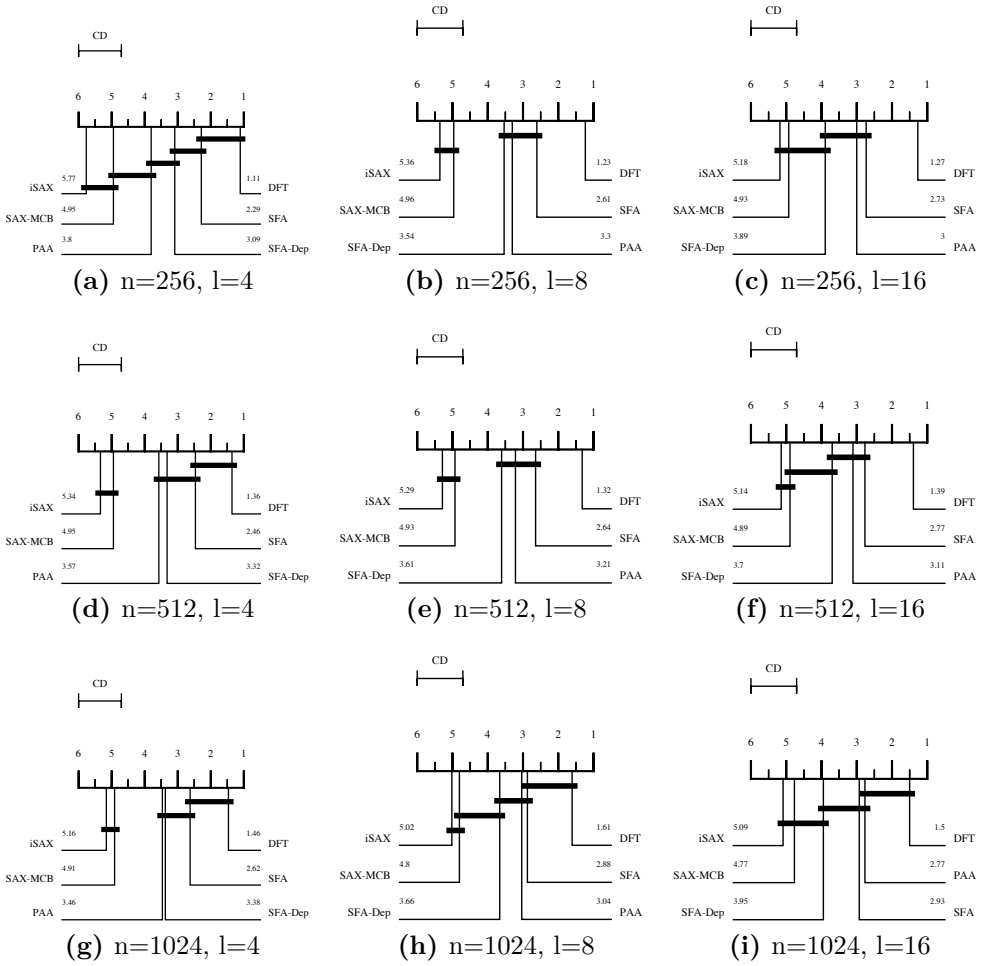


Figure 3.13 – Critical difference diagrams over the average ranks of the *pruning power* (lower is better) for different quantization techniques on the 29 real datasets for time series lengths n=[256,512,1024], l=[4,8,16] values, and c=256 symbols.

To show the impact of the MCB quantization, we compared the influence of quantization techniques on the pruning power. We measured the *pruning power* on the 29 datasets using different techniques:

1. Without quantization: DFT and PAA.
2. With quantization:
   (a) Equi-depth binning:

i. *SFA-Dep* using equi-depth binning of the Fourier value distribution. It uses the same equi-depth bins for all values. This is in concept similar to the SAX/iSAX quantization approach.

ii. *iSAX/SAX* using PAA and equi-depth binning of the *Gaussian distribution* as published in [46]. It uses the same equi-depth bins for all mean values.

(b) MCB binning:

i. *SFA* using MCB quantization.

ii. *SAX-MCB* using PAA, and our MCB quantization. Thus, it uses different equi-depth bins for each mean value like SFA.

We used the same setup as in the previous experiment. Figure 3.13 shows the critical difference diagrams over the average ranks based on the pruning power.

- iSAX/SAX performs worst in the experiments. This is a result of the use of PAA for quantization which is significantly worse than DTW.
- *SAX-MCB* only slightly improves over *SAX*. The reason has been stated in [46, 47]: in the time domain the distribution for each mean value will be the same for the subsequences, due to all points being shifted over all positions of the time series. Thus using different quantization intervals for each PAA value does not improve the accuracy but using data adaptive quantization intervals improves it slightly.
- *SFA* with *MCB* shows a very similar pruning power as DFT.
- *SFA-Dep* is up to 1 rank worse than *SFA*. This confirms that the MCB quantization guarantees a very tight approximation of the DFT, and as such SFA performs better than many other dimensionality reductions.

Implications: We conclude that differences in the quantization technique have a smaller impact on the *pruning power* than the dimensionality reduction technique. SFA using MCB performs better than SFA-Dep and SAX. SFA is favorable over numeric dimensionality reduction techniques such as DFT because of its smaller memory footprint and its symbolic representation (Chapter 3.3.8).

3.5.2 Indexing High Dimensional Datasets

In these experiments we benchmark SAMs and focus on two parameters:

1. Values/word length l: This represents the length of the time series after dimensionality reduction, and as such the indexable length by each SAM (Curse of Dimensionality). Exact approximations (larger values/word lengths) are desirable.
2. Time series length n: Larger n yield in a larger compression ratio.

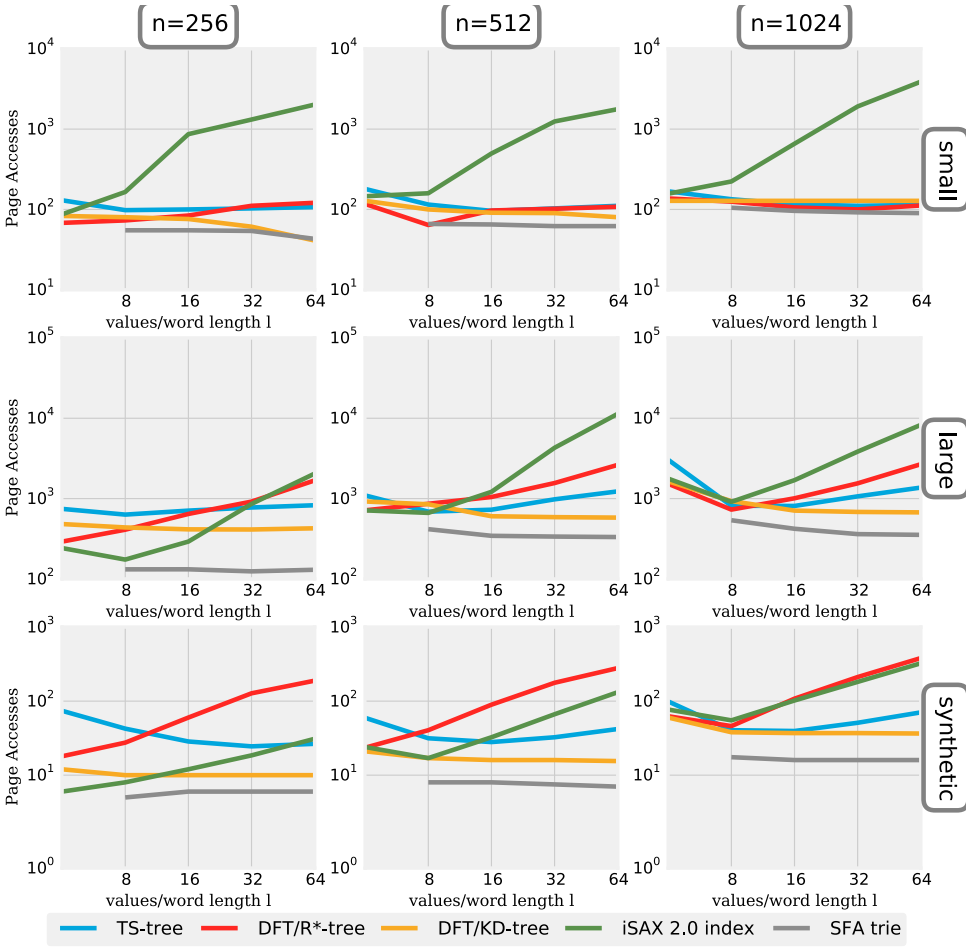


Figure 3.14 – Median page accesses for the real world and synthetic datasets partitioned into three groups: Small, Large, and Synthetic.

Setup: We evaluate the performance of 10-NN queries on the 10 synthetic random walk datasets and the 29 real world datasets presented in Table 3.3 using:

1. The SFA trie,
2. The iSAX 2.0(+) index [20, 21],
3. The Time-Series-tree (TS-tree) [10],
4. DFT with the R*-tree [36, 16].
5. DFT with the KD-tree [18].

We chose DFT as the representative for the numeric dimensionality reduction techniques, as DFT has the best *pruning power* in all the experiments. The R*-tree is the typical

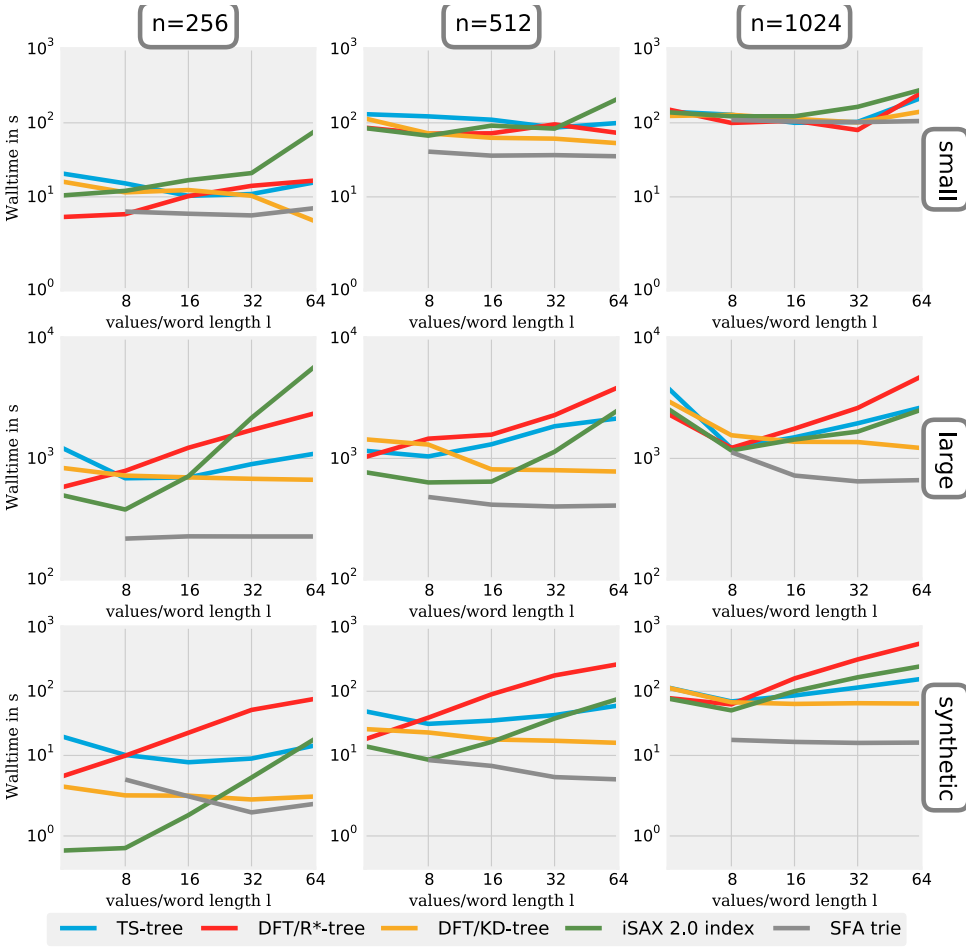


Figure 3.15 – Median wall-clock time for the real world and synthetic datasets partitioned into three groups: Small, Large, and Synthetic.

SAM used in time series literature. We performed 1000 10-NN queries on all datasets with subsequence/query lengths of $n = \{256, 512, 1024\}$.

The synthetic datasets contain 100000 subsequences each. The real world datasets contain 2.500 (winding) up to 5.4 million (StarlightCurve) subsequences. For the experiments we partitioned the real world datasets into two groups *small* and *large*, with 16 datasets with less than 100000 subsequences and 11 real datasets with more than 100000 subsequences.

Each SAM was configured to have a maximum of 100 time series per leaf node (same as in [82]), before it is split. A leaf node access accounts for one random page access even if the leaf node spans more than one disk block or if two leaf nodes are stored in successive blocks on hard disk.

We used the best parameter settings according to the authors of each of the SAMs:

- For the iSAX 2.0(+) index we set the base cardinality (alphabet size) $b = 4$, $c = 256$

symbols, word length $l = 8$, and a threshold $th = 100$. We benchmarked the optimized node splitting policy presented in [20, 21].

- For the R*-tree we set the $fillfactor = 100$ and $p = 0.80$.
- For the SFA trie we set the fanout to $c = 8$ (alphabet size) and the threshold $th = 100$.
- For the TS-tree we set the base cardinality to $b = 4$, $order = 50$ and the threshold $th = 100$.
- For the KD-tree we use the maximum spread to select the optimal dimension for splits when a leaf overflows. This is the standard splitting rule.

Impact of Values/Word Length l: The length of the approximation is a critical parameter, as a too low (all time series map to the same points) or too high (Curse of Dimensionality) dimensionality will lead to a rapid degeneration of the search performance using the index up to a point where a sequential scan of the dataset is faster. We have to decide on a dimensionality, before building the index structure.

In this experiment we test the scalability in terms of the length of the approximations, i.e., varying the word length which correlates with the indexable dimensionality. Figure 3.15 shows the scalability in terms of the *median wall-clock time* and Figure 3.14 shows the *median page accesses* to answer 1000 10-NN queries. The *plot columns* represent the time series lengths n. The *plot rows* represent the different datasets small, large and synthetic. The minimum of each line shows the best choosable configuration for each SAM.

Both measurements (wall-clock time and page accesses) show the same trends:

- All index structures degenerate when increasing l from 4 up to 64 except for the SFA trie and KD-tree using DFT, which scale with the increase of dimensions.
- The SFA trie scales to 64 indexed dimensions without degeneration. $20 - 32$ dimensions seem to be a good trade-off between storage space and performance. This equals $60 - 96$ bits for a single SFA word.
- Comparing the optimal number of indexed dimensions for the iSAX 2.0 index (8), the R*-tree $(4 - 8)$, the KD-tree $(8 - 32)$, the TS-tree $(8 - 16)$, and the SFA trie $(20 - 32)$, the performance of the SFA trie is among the best on almost all datasets.

Implications: The SFA trie has the lowest page accesses and wall-clock times, and it is not sensitive to the choice of the length (dimensionality) of the approximations l. In the SFA trie an increasing dimensionality of the approximations causes dense leaf nodes to split, but the general structure of the trie remains unaffected as opposed to the R*-tree or the iSAX 2.0 index. This is why the SFA trie improves when increasing the length of the approximation, whereas others degenerate for a too high dimensionality.

3.5.3 Indexing One Billion Time Series

In this experiment we focus on the scalability in terms of the size of the dataset N, i.e., the number of time series it contains.

Setup: iSAX and SFA qualify for indexing large-sized datasets as both are symbolic representations and offer an at least 8-fold lower memory footprint than numerical dimensionality reductions. For this experiment, we indexed 10^6 (1 M) to 10^9 (1000 M) synthetic time series with a length of 256, resulting in 2GB to 2TB of raw time series data. We used the same data generation process as described in iSAX 2.0 [20]: Each data point was generated using standard normal distribution N(0,1) and the recursive function: $x_{i+1} = x_i + N(0,1)$. We used the best parameter settings according to the authors of each of the SAMs:

- For the iSAX 2.0 index we set the word length $l = 8$, the base cardinality $b = 4$, and a leaf threshold $th = 10000$. We used the optimized node splitting policy presented in [20, 21].
- For the SFA trie we set $l = 20$, alphabet size $c = 8$, and a leaf threshold $th = 10000$.

Each SAM can store up to 10000 time series before it is split. A leaf node access accounts for one page access even if the leaf node spans more than one disk block or if two leaf nodes are stored in successive blocks on the hard disk.

Index Size & Leaf Node Occupancy: In this experiment we measure the size of the SAM in terms of:

1. The total number of nodes (including leaf nodes),
2. The total number of leaf nodes,
3. The fill-factor of the leaf nodes, and
4. The size of the index excluding the raw time series data.

Figure 3.16 shows that the SFA trie is a more compact representation than the iSAX 2.0 index and requires up to 2.4 times less nodes and up to 2.7 times less leaf nodes to index the data. This is equal to up to a 2.7 times higher (better) fill-factor of the leaf nodes. However, the SFA trie needs up to 50% more main memory compared to the iSAX 2.0 index (excluding the raw time series data) on the largest dataset size.

Implication: These results confirm that the SFA trie qualifies for indexing large datasets in terms of the memory footprint.

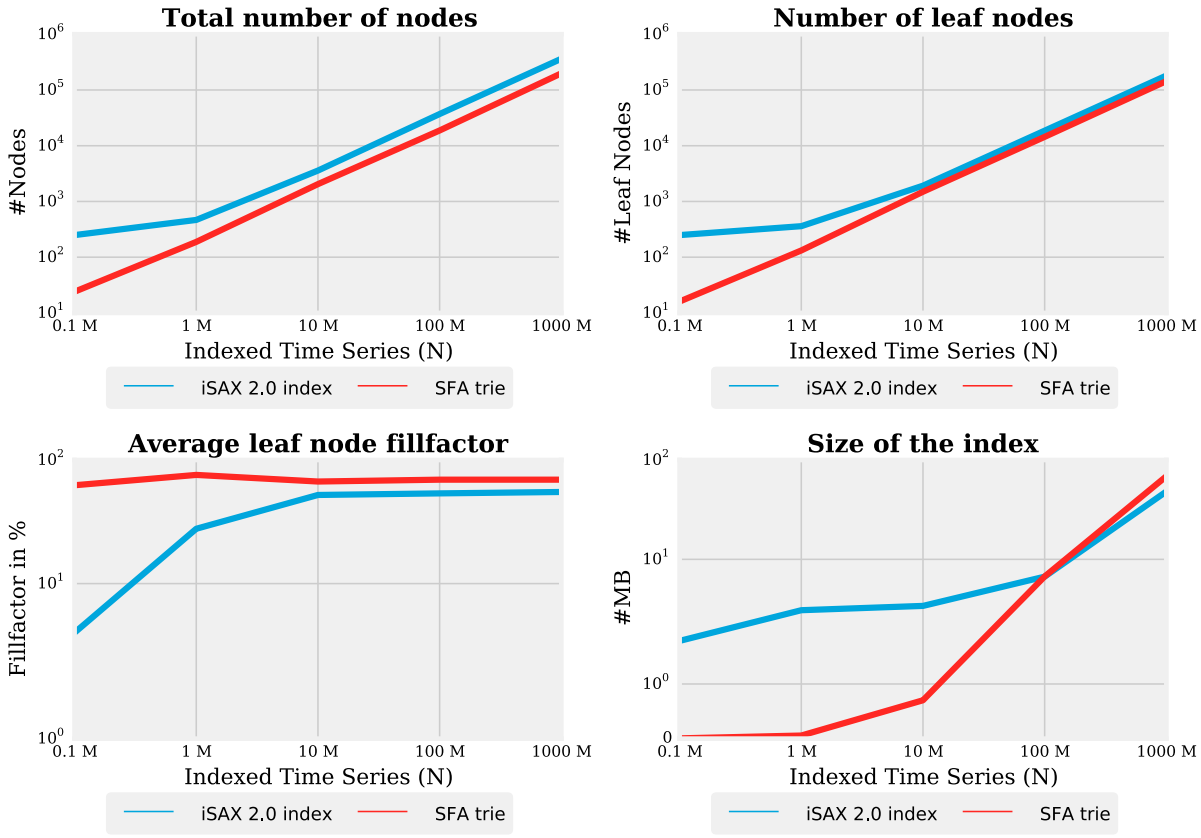


Figure 3.16 – Comparison of index size and leaf node occupancy with a varying size of the synthetic dataset.

Similarity Search: To benchmark the SAMs, we compared the iSAX 2.0 index and the SFA trie in terms of average leaf node accesses which are an objective measure for *disk page accesses*, and the average *wall-clock time* that is the time spent for a similarity query including disk page accesses. We performed 100 5-NN queries each and averaged the results.

This experiment (Figure 3.17) demonstrates that the SFA trie requires less disk accesses and less wall time than the iSAX 2.0 index to answer a similarity query. On average the SFA trie requires up to 50% less wall time and up 3 times less leaf node accesses.

A single 5-NN query on the 1000M time series dataset took on average 41 minutes for the SFA trie and 63 minutes for the iSAX 2.0 index. The iSAX 2.0+ index [21] introduces a bulk loading mechanism, which does not have any effects on similarity query performance.

Implication: Note that the use of N(0,1) for data generation favors iSAX, as iSAX quantization is based on N(0,1). Still, the SFA trie requires less page accesses and wall time than the iSAX 2.0 index.

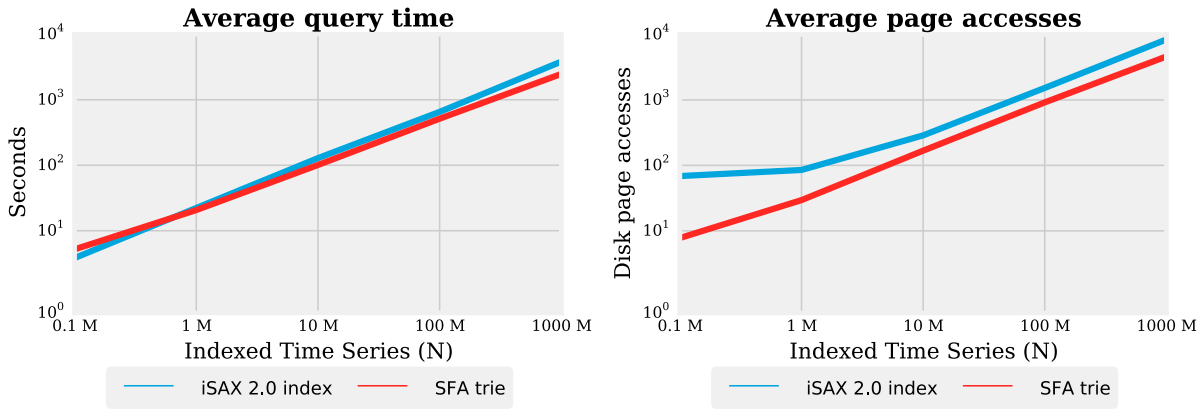


Figure 3.17 – Average time and average page accesses to answer a 5-NN query with a varying size of the synthetic dataset.

3.6 Summary

We introduced a novel symbolic representation of time series called Symbolic Fourier Approximation (SFA). SFA is based on the quantization of Fourier transformed time series. As part of SFA we introduced a novel quantization technique called multiple coefficient binning (MCB). As DFT has a statistically significant better pruning power than other dimensionality reduction techniques, SFA provides a better pruning power than most other dimensionality reduction techniques including the symbolic representation SAX/iSAX. SFA is a dimensionality reduction technique that provides (a) noise removal using low-pass filtering, (b) a string representation that allows for string matching algorithms to be applied, and (c) the frequency domain nature. It is the frequency domain nature that makes it unique among the symbolic time series representations. Dynamically adding or removing Fourier coefficients to adapt the degree of approximation without recalculating the Fourier transform is at the core of all presented algorithms in this thesis.

We have introduced the SFA trie for exact time series similarity search. The SFA trie exploits the frequency domain nature of SFA by approximation of a time series using a high dimensionality and a variable prefix length for indexing. With a variable prefix length it is possible to add detail on the fly and distinguish time series which have similar approximations. This leads to an improved similarity query performance. The SFA trie is tailored for a variable prefix length as it grows in depth rather than width when increasing the length of similar approximations, which postpones the effects of the Curse of Dimensionality. In our experiments the SFA trie is the best index structure in terms of page accesses and wall-clock time on real and synthetic datasets, and allows for indexing terabyte-sized time series datasets in main memory.

Chapter 4

Shotgun Distance: Towards Alignment-free Time Series Data Analytics

The Shotgun distance is the first time series model presented in this thesis (compare Figure 1.2). This chapter gives a detailed description of the Shotgun distance. It is based on and contains text passages from my three publications [71, 69, 58].

4.1 Introduction

Empirical evaluation suggests that distance measures like the Euclidean distance (ED) or dynamic time warping (DTW) are hard to beat [12, 29, 13]. However, these have some known shortcomings: The ED neither provides horizontal alignment nor supports variable length time series. DTW provides warping invariance, which is a peak-to-peak and valley-to-valley alignment of two time series and typically fails if there is a variable number of peaks and valleys. Figure 4.1 shows a dendrogram of a hierarchical clustering of a synthetic dataset. It consists of three types of shapes that have variable lengths and phase shifts. Even though the dataset is very simple, the distinguishing powers of both the ED and DTW distance measures are rather disappointing. The ED fails to separate the shapes as it neither supports horizontal alignment nor variable lengths. Neither does DTW result in a satisfying clustering as it fails to separate the triangles from the sine waves. Our Shotgun distance clusters all shapes correctly. This example illustrates just a portion of the difficulties arising from time series similarity.

In general, several sources of invariance like *amplitude/offset, warping, phase, uniform scaling, occlusion, and complexity* have been identified (Chapter 2.2). Both, ED and DTW calculate the distance between two entire time series to determine their similarity. To make these applicable a significant amount of time and effort have to be spent by a domain expert to filter the data and extract equivalent-length, equal scale, and aligned characteristic patterns. The UCR time series benchmark datasets contain many such datasets [42]. In our synthetic example the shapes have to be aligned and trimmed to

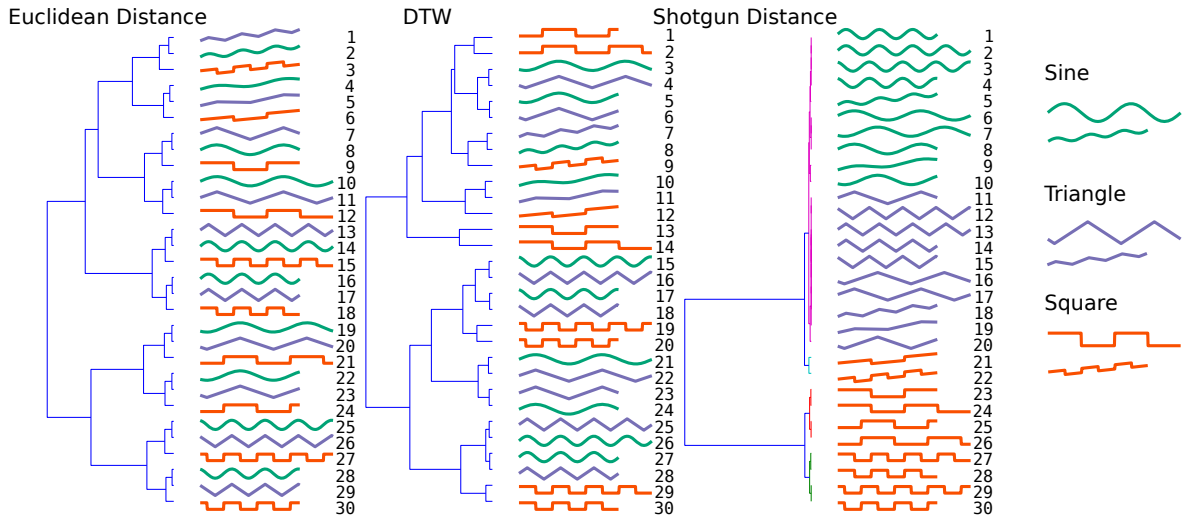


Figure 4.1 – A dendrogram of a hierarchical clustering of a synthetic dataset based on three similarity measures. There are three types of curves: Sine, Square, Triangle.

equivalent-length for the ED and DTW distance measures to give meaningful results. Human assistance significantly eases the subsequent data mining task both in terms of the execution time and the complexity of the algorithm. However, human assistance is often too time consuming, expensive [93, 55], and does not scale to large data volumes. Only few time series algorithms exist that deal with the data 'as is'. These algorithms are based on searching for similarities in substructures (compare Chapter 2.2). The idea is to deliberately ignore some data, by extracting local, representative subsequences from a time series. As traditional data mining algorithms are not easily applicable to raw datasets, international competitions were staged like identifying whale calls [2], human walking motions [2], or monitoring flying insects [1].

This thesis introduces a simple and novel similarity measure for time series similarity data analytics. The Shotgun distance breaks a query time series into subsequences and vertically aligns and horizontally scales each subsequence prior to calculating the similarity to a sample time series. This simplifies the need for human preprocessing of a time series to extract and align these characteristic segments (this is what we refer to as *alignment-free*). Figure 4.2 shows the Shotgun distance model. In this example, a query time series is segmented into disjoint windows of length 120, which is roughly equal to one gait cycle. For each query window the similarity is separately calculated by shifting the window along the sample, searching for the offset that minimizes the Euclidean distance.

The contributions of the Shotgun distance are as follows:

- The Shotgun distance aims at alignment-free time series similarity. That is, the datasets do not have to be preprocessed by domain experts for equivalent-length, or approximately aligned substructures.

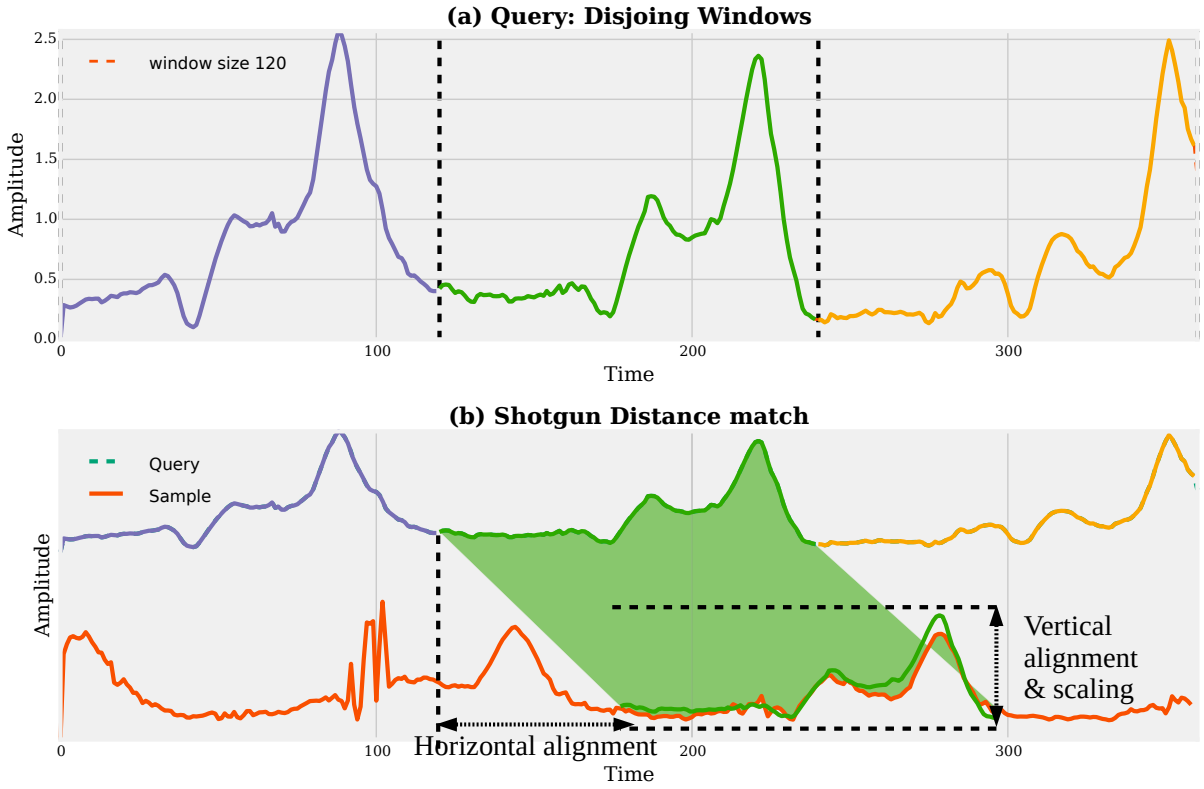


Figure 4.2 – Shotgun distance consists of subsequence extraction, horizontal and vertical alignment, and scaling.

- We introduce the Shotgun distance that is based on vertical scaling and horizontal alignment in Chapter 4.3.
- We present the Shotgun ensemble classifier which is an ensemble of 1-NN Shotgun classifiers utilizing the Shotgun distance at multiple subsequences lengths in Chapter 4.4.3.
- Two early abandoning strategies are presented to stop computations of unpromising candidates. This significantly reduces the computational complexity by one order of magnitude in Chapter 4.5.
- We present case studies on not preprocessed (unaligned) time series data for hierarchical clustering and classification in Chapter 4.6. The Shotgun ensemble classifier is more accurate than state of the art on the case studies and the UCR time series benchmark datasets.

4.2 Background and Related Work

Time series similarity search is a complex task for a computer. It is non trivial to extract a general statistical model from time series as the time series may show varying statistical properties with time (non-stationary). Classic machine learning algorithms applied to the raw time series degenerate due to the high dimensionality of the time series and noise [43]. Approaches can be characterized by "*Shape-based"* and *"Structure-based*" (compare Chapter 2.2):

1. *Shape-based:* These are based on the whole time series and perform a point-wise comparison. Shape-based techniques include the 1-NN Euclidean Distance (ED), or the 1-NN DTW [62, 65] with a consensus that DTW is among the best time series similarity measures [49, 13, 29]. Typically, shape-based techniques work well for short time series but fail for long time series [68, 38], such as the ones we consider in this chapter.

2. *Structure-based*: These transform the data into an alternative data space to make existing data mining algorithms applicable [12, 38, 48, 52, 63, 91, 88]. The techniques are based on data mining algorithms such as 1-NN, SVMs, decision trees, or random forests in combination with feature extraction from the raw time series. Feature extraction techniques include the dimensionality reduction techniques (Chapter 2.3) or Shapelets [92, 52, 90, 91, 63], which are representative variable-length subsequences of a time series. By transforming time series data into an alternative data space, the accuracy of classification algorithms can be significantly improved [12]. However, these authors fail to show a significant improvement over 1-NN DTW which is the benchmark algorithm for time series analysis. Shapelet classifiers [52, 63, 91] are used in combination with a decision tree. The Shapelets are stored within the nodes of the tree and a distance threshold is used for branching. One Shapelet algorithm explicitly deals with the classification on raw time series data [38].

Our Shotgun classifier is a structure-based method that is inspired by Shotgun sequencing introduced to find an alignment of two DNA or protein sequences [86]. Shotgun sequencing was used to find the horizontal displacements of steel coils [50]. To find the horizontal displacement the authors use the median on the differences of the calculated starting positions for every pair of subsequences. In contrast, our Shotgun distance accumulates the Euclidean distances of the subsequences.

4.3 Shotgun Distance: An Alignment-free Similarity Model

4.3.1 Motivation

The utility of the Shotgun distance is tied to the observation that a multitude of signals contains characteristic/distinctive patterns. Consider human walking motions [26] as

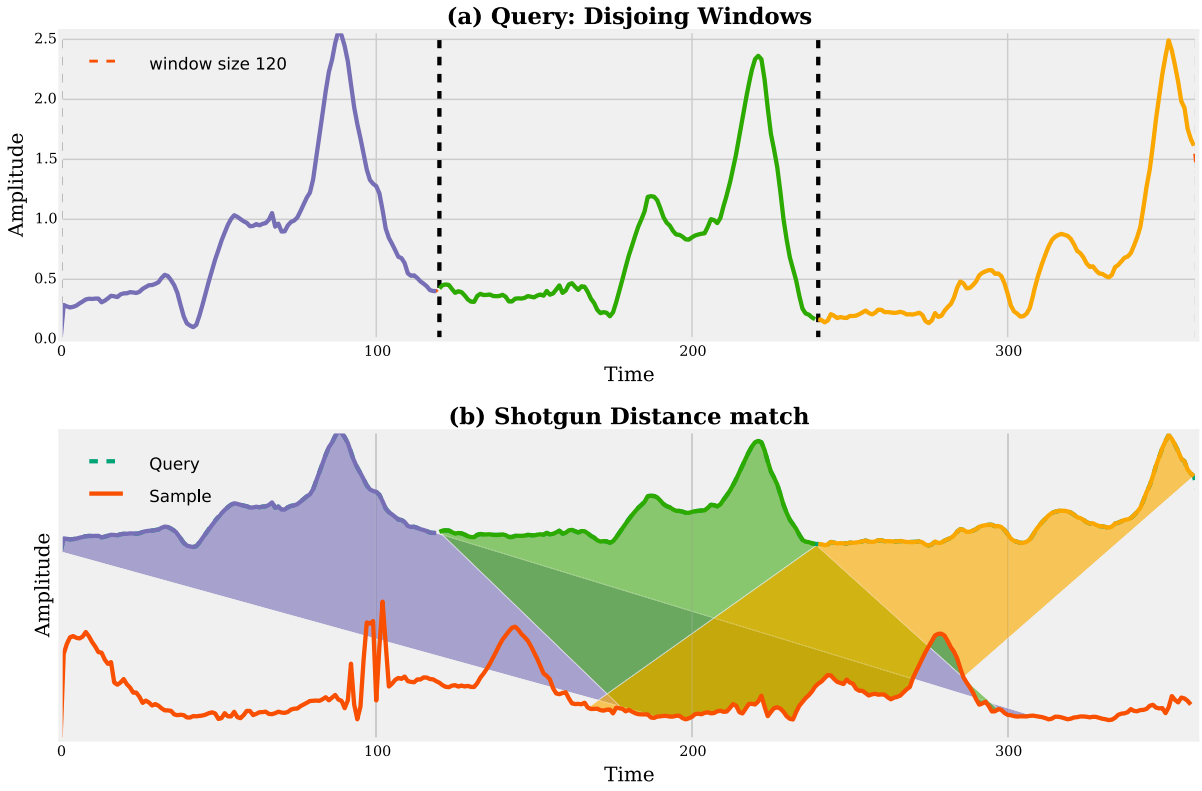


Figure 4.3 – Matching the gait cycles in a query to the sample is complicated due to different amplitudes, phase-shifts, variable lengths, noise and a variable number of gait cycles.

a concrete example. The data were captured by recording the z-axis accelerometer values of either the right or the left toe. The difficulties in this dataset result from variable-length gait cycles, gait styles and pace of different subjects throughout different activities. Figure 4.3 (top) illustrates the walking motions of a subject. The query motion is composed of 3 gait cycles (top). The sample (bottom) is a motion that contains only one full gait cycle whereas the others are distorted by noise. In general, classifying walking motions is difficult, as the samples may not (a) be aligned, (b) have variable length, scale or number of gait cycles, and (c) the recordings are typically distorted by noise.

A common approach in literature is to preprocess the data by a domain expert to extract equivalent-length, approximately aligned substructures, and filter noise. The UCR time series benchmark datasets contain many such preprocessed datasets [42]. Searching for similarities in substructures (compare Chapter 2.2) rather than matching the time series as a whole is the basic idea of our Shotgun distance to deal with the alignment-free similarity of long time series. Shotgun distance reduces the need for the cost-ineffective preprocessing by performing vertical alignment and horizontal scaling between the query and the sample time series. As such we call it an *alignment-free* time series similarity measure. Figure 4.3 (bottom) illustrates the result of the Shotgun distance matching.

The 3 query gait cycles match to the one full gait cycle in the sample.

The quality of the Shotgun distance is subject to two parameters (Figure 4.2):

1. *Horizontal alignment* using the *window length* $w \,\epsilon\, \mathbb{N}$: An Integer parameter that is limited by the length of the longest query. The *window length* parameter depends on the length of the characteristic patterns in the dataset. It regulates how much information on the ordering of the values within the time series is incorporated into the matching-process: For long window lengths the whole query will be treated as a single pattern (similar to the Euclidean distance). Most of the time this happens with signals that were preprocessed. In contrast, the human walking motions contain multiple gait cycles. Aligning any gait cycle in the query to any gait cycle in the sample is equivalent. Thus, the ordering information is less relevant, resulting in a window length that is roughly equal to one gait cycle.

2. *Vertical alignment* using the $mean \,\epsilon\, [true, false]$: A Boolean parameter that defines if the mean value should be subtracted prior to the distance calculations. The mean value gives offset invariance (Chapter 2.2). The standard deviation is always normed to 1 to obtain amplitude invariance. For example, heart beats have to be compared using a common baseline but the pitch of a bird sound can be significant for the species. Surprisingly, up to now in time series literature mean normalization has not been considered as a parameter.

4.3.2 The Shotgun Distance Model

The Shotgun distance is based on windowing (see Equation 2.5). To *vertically* align two samples, the query window and the sample window can be z-normalized by subtracting the mean and dividing them by the standard deviation (see Chapter 2.2). The normed windows $\hat{\omega}$ are given by:

$$\hat{\omega}(T, w, step) = z\_norms(windows(T, w, step)) \tag{4.1}$$

$$= \{z\_norm(S) \mid S \,\epsilon\, windows(T, w, step)\} \tag{4.2}$$

The mean normalization as part of the z-norm is treated as a parameter of the Shotgun distance model and can be enabled or disabled. The Shotgun distance minimizes the Euclidean distance between each disjoint window in the query Q and the sliding windows in a sample S: Each gait cycle is slid along the longer walking motion to find the best matching positions in terms of minimizing the Euclidean distance.

Definition 12. *Shotgun Distance*: The Shotgun distance $D_{shotgun}(Q, S)$ between a query Q and a sample C is given by adding up the minimal Euclidean distance $D_{ED}(Q_a, S)$ between each disjoint query window $Q_a \,\epsilon\, \hat{\omega}(Q, w, w)$ and each offset in C, represented by

Algorithm 4.1 The Shotgun distance.

```
1  double ShotgunDistance(TimeSeries query, TimeSeries sample, int w, bool mean_norm)
2    double totalDist = 0.0
3    // for each disjoint query window
4    for TimeSeries q in disjoint_windows(query, w, mean_norm)
5      double qDist = MAX_VALUE
6      // find the position that minimizes the Euclidean distance
7      for TimeSeries s in sliding_windows(sample, w, mean_norm)
8        qDist = min(qDist,D_ED(q,s))
9      totalDist += qDist
10   return totalDist
```

the sliding windows $S \, \epsilon \, \hat{\omega}(C, w, 1)$:

$$D_{shotgun}(Q, C) = \sum_{a=1}^{len(\hat{\omega}(Q,w,w))} \min \{D_{ED}(Q_a, S) \mid S \, \epsilon \, \hat{\omega}(C, w, 1)\} \tag{4.3}$$

$$for Q_a \, \epsilon \, \hat{\omega}(Q, w, w) \tag{4.4}$$

This definition resembles the extraction of characteristic patterns, i.e., the gait cycles, and the scaling and alignment of the patterns. The latter provides invariance to the time ordering of the patterns and allows for comparing variable length time series. The Shotgun distance is equal to the Euclidean distance for $w = n$.

The Shotgun distance is not a *distance metric* (compare Chapter 2.2) as:

- It does not satisfy the *symmetry condition* and
- It does not satisfy the *triangle inequality*: $\exists T \mid D_{shotgun}(Q, C) > D_{shotgun}(Q, T) + D_{shotgun}(T, C)$.

This is a result of the use of disjoint query windows and sliding sample windows. As a consequence the Shotgun distance does not allow for exact indexing and if C is the nearest neighbor of Q, Q does not have to be the nearest neighbor of C.

The Shotgun distance webpage [70] contains a video that illustrates the Shotgun distance.

4.3.3 The Shotgun Distance Algorithm

The Shotgun distance (Algorithm 4.1) makes use of the Euclidean distance, and can be tuned by the two parameters (a) *window length* w and (b) *mean normalization mean_norm*. The standard deviation of q and s is always normed to 1 regardless of mean_norm. The algorithm first splits the query into disjoint windows (line 4) and searches for the position in the sample that minimizes the Euclidean distance (lines 7–8). Finally, the minimal Euclidean distances are added up for each query window (line 9).

Algorithm 4.2 The Shotgun classifier.

```
String predict(TimeSeries query, TimeSeries[] samples, int w, bool mean_norm)
  (double dist, TimeSeries nn) = (MAX_VALUE, NULL)
  for TimeSeries sample in samples
    double D = ShotgunDistance(query, sample, w, mean_norm)
    if D < dist
      (dist, nn) = (D, sample)
  return label(nn)
```

Computational Complexity: The computational complexity is quadratic in the length of the time series Q and C: For each query window, all sample windows are iterated and the Euclidean distance for each pair of windows is calculated. There are $\frac{|Q|}{w}$ disjoint query windows and $|C| - w + 1$ sliding windows for window length w:

$$T(\text{Shotgun Distance}) \quad \epsilon \quad O\left(\underbrace{\frac{|Q|}{w}}_{\text{disjoint windows}} \cdot w \cdot \underbrace{(|C| - w + 1)}_{\text{sliding windows}}\right) \tag{4.5}$$

$$\Rightarrow \quad O\left(n^2 - nw\right), \qquad \text{for } n = \max(|Q|, |C|) \tag{4.6}$$

Note that for large window lengths $w \sim n$ this complexity is close to linear in n (like the Euclidean distance). For small window lengths $w \ll n$ the complexity is quadratic in n (like DTW).

4.4 Time Series Classification

Classification describes the task of assigning a label to an unlabeled time series Q. Figure 4.4 illustrates this classification task. The 1-NN Shotgun classifier is a 1-NN classifier. Thus, it uses a set of train samples DS as model and performs a 1-NN search to find the label of Q. The query is assigned to the class of the time series C that minimizes the Shotgun distance:

$$label(Q) = \underset{T \epsilon DS}{argmin}(D_{shotgun}(Q, C))$$

The two parameters of the Shotgun distance model are part of the classification model. As the labels of the train time series have to be known beforehand, this is referred to as supervised learning.

4.4.1 The Shotgun Classifier Algorithm

Our Shotgun classifier is based on 1-NN classification and the Shotgun distance. We chose to use 1-NN classification as it is very robust and doesn't introduce any additional parameters for model training [29].

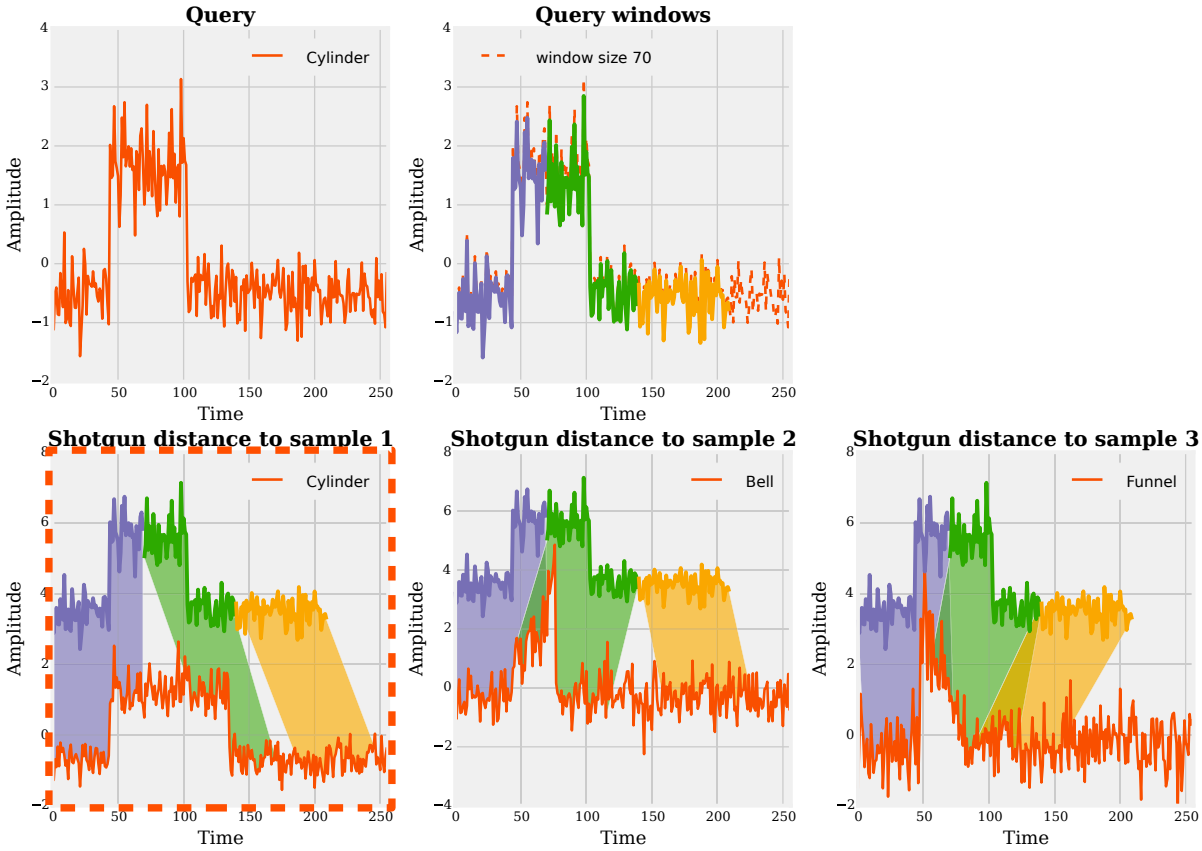


Figure 4.4 – 1-Nearest-Neighbor Shotgun classifier: It labels a query by the label of the 1-NN to it. The best match is highlighted.

Prediction (Algorithm 4.2): Given the query, the *predict*-method searches for the 1-NN to the query within the set of samples (lines 3–6). Finally, the query is labeled by the class label of the 1-NN *nn* (line 7).

Parallel Execution (Algorithm 4.2): We do not parallelize a single *predict*-method-call. Instead, when running multiple queries, each 1-NN query (*predict*-method-call) runs in a separate thread. This workload is embarrassingly parallel.

Fit (Algorithm 4.3): The *fit*-method has to find the optimal set of parameters for *window length* w and the *mean normalization* *mean_norm*. The *mean_norm*-parameter is a Boolean parameter, which is constant for a whole dataset as opposed to setting it per sample or window. The algorithm performs a grid-search over the parameter space using leave-one-out cross-validation on the train samples (lines 6–8). This results in the parameters that maximize the accuracy on the train samples. All accuracies for all window lengths starting from the $maxLen$ (the length of the longest time series) down to

Algorithm 4.3 Training the Shotgun classifier.

```
1  [(int,int)] fit(TimeSeries[] samples, bool mean_norm)
2    [(int,int)] scores = []
3    // search for best window lengths
4    for int w = maxLen down to minLen // in parallel
5      int correct = 0
6      for TimeSeries query in samples // leave-one-out, in parallel
7        String nnLabel = predict(query,samples\{query},w,mean_norm)
8        if (nnLabel==query.label) correct++
9      // store scores for each window length
10     scores.push((correct, w))
11   return scores
```

minLen (line 10) are recorded. We will use these for our ensemble classifier introduced in Chapter 4.4.3.

Parallel Execution (Algorithm 4.3): The training of the classifier is well-suited for parallel execution: There is no data-dependency among window lengths in line 4. A nested embarrassingly parallel region is given in lines 6ff for each 1-NN search.

4.4.2 Computational Complexity

Prediction: The computational complexity for classification is given by a 1-NN search over the N train time series using the Shotgun distance:

$$\begin{aligned} T\left(\text{Shotgun Predict}\right) &\epsilon O\left(N \cdot T(\text{ShotgunDistance})\right) && (4.7)\\ &\epsilon O\left(N \cdot (n^2 - nw)\right) && (4.8)\end{aligned}$$

It has the same computational complexity as the 1-NN DTW algorithm with $O(Nn^2)$. We postpone a comparison of the computational complexity of the rivaling methods to Chapter 6.

Fit: To find the optimal window length w it takes $O(n)$ executions of leave-one-out cross-validation on the train samples. Leave-one-out cross-validation has a quadratic computational complexity in the number of samples N:

$$\begin{aligned} T\left(\text{Shotgun Fit}\right) &\epsilon O\left(\sum_{w=minLen}^{n} N \cdot T\left(\text{Shotgun Predict}\right)\right) && (4.9)\\ &\epsilon O\left(\sum_{w=minLen}^{n} N^2 \cdot (n^2 - nw)\right) && (4.10)\\ &= O\left(N^2 \cdot \frac{n^3 - n^2}{2}\right) && (4.11)\end{aligned}$$

If the length of the characteristic patterns within a dataset is known ahead of time, it is trivial to reduce the computational complexity to $O(N^2 n^2)$ by testing only window lengths that are roughly equal to the pattern length. We postpone a comparison of the computational complexity of the rivaling methods to Chapter 6.

4.4.3 Shotgun Ensemble Classifier Algorithm

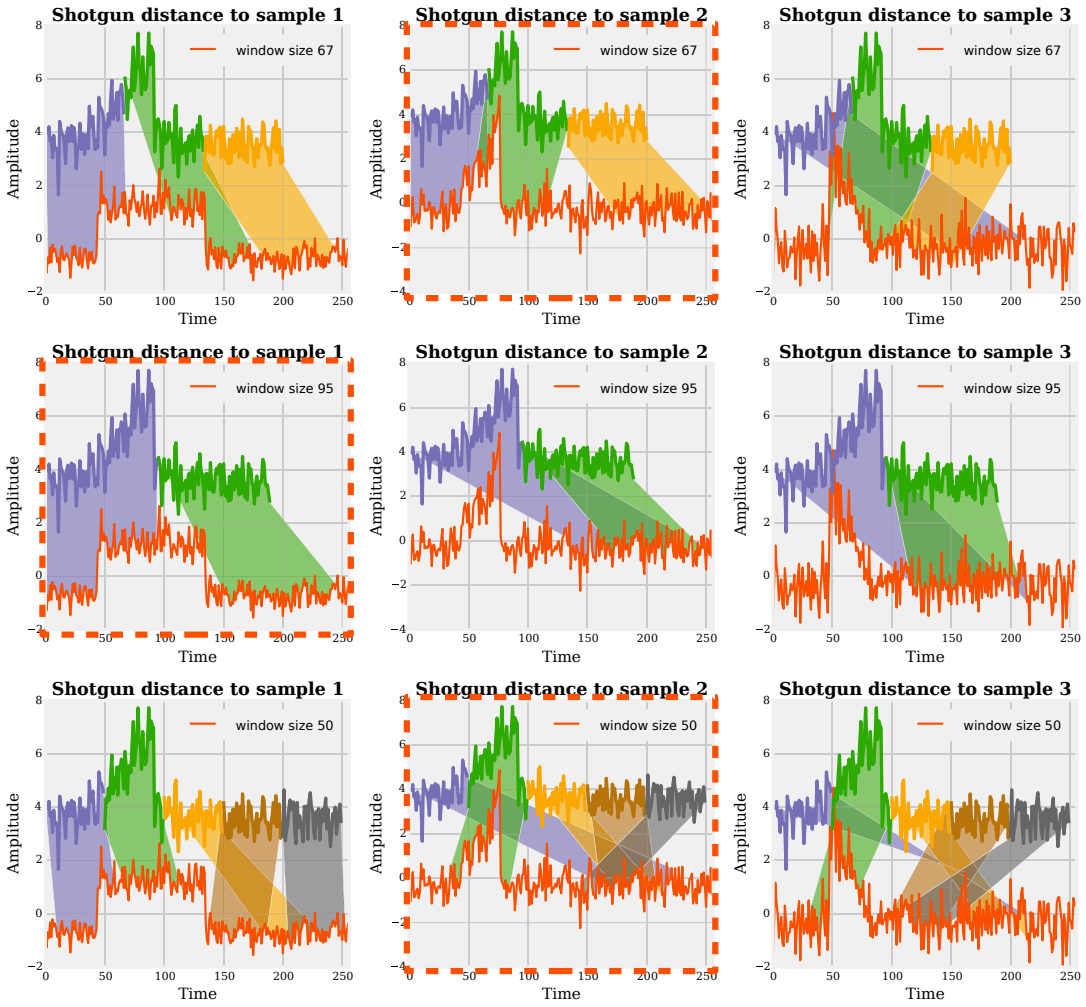


Figure 4.5 – In this example the Shotgun ensemble classifier uses three window lengths: 67, 95 and 106. The 1-NNs are: Bell, Cylinder, Bell. A majority count decides for the best match: Bell.

By intuition every dataset is composed of substructures of varying window lengths caused by varying lengths of characteristic shapes like different walking styles, heart beats, or duration of vocals. To cope with these datasets the Shotgun classifier algorithm is

Algorithm 4.4 The Shotgun ensemble classifier.

```
1  String predictEnsemble(TimeSeries query, TimeSeries[] samples, [(int,int)] scores, bool
       mean_norm, double factor)
2    // stores for each window length a label
3    String[] windowLabels = []
4    int bestScore = max([ correct | ( correct,_) in scores ])
5    // determine the label for each window length
6    for (correct, w) in scores // in parallel
7      if (correct > bestScore*factor)
8        windowLabels[len] = predict(query, samples, w, mean_norm)
9    return most frequent label from windowLabels
```

extended to support multiple window lengths, thus making it an ensemble technique and further adding to the robustness of the classifier.

Figure 4.5 illustrates the basic rationale of the Shotgun ensemble classifier for walking motions using three different window lengths: 67, 95 and 106. The query is split into disjoint windows of corresponding lengths and the 1-NN is searched within the three samples. Each window length results in a 1-NN. A majority count among *Bell, Cylinder, Bell* leads to the class *Bell*.

Algorithm 4.2 implements the Shotgun classifier for one fixed window length. The Shotgun ensemble classifier in Algorithm 4.4 extends this algorithm.

Prediction (Algorithm 4.4): The *fit*-method in Algorithm 4.3 returns a list of scores (accuracies), each one resulting from a different window length on the train samples. The Shotgun ensemble classifier (Algorithm 4.4) performs classification using the best window lengths from this list of scores. The highest train accuracy on the train samples is given by *bestScore* (line 4). Using a constant parameter $factor \epsilon (0.01, 1]$ and this *bestScore*, the optimal window lengths are given by:

$$correct > bestScore \cdot factor$$

If *factor* is set to 1, only the best window length is used. If *factor* is set to 0.01, (almost) all window lengths will used. In our experiments, factors in between $[0.92, 1.0]$ were best throughout the datasets.

For each window length the predicted label is recorded (line 8). Finally, a majority count is performed, i.e., the most frequent class label is chosen from these labels (line 9).

While it might seem that the Shotgun ensemble classifier adds a third parameter *factor*, the training of the Shotgun ensemble classifier depends solely on the two *factor* and *mean* parameters (and not on the window length). The Shotgun ensemble classifier model is derived from these two parameters using the *fit*-method, which returns the list of window scores. These scores serve as the model and are used to predict the label of an unlabeled query.

Parallel Execution (Algorithm 4.4): There are no data-dependencies among the window lengths (line 6). We run each window length in a separate thread/process.

Algorithm 4.5 Early abandoning Euclidean distance computations.

```
1 double D_ED(TimeSeries query, TimeSeries sample, double bestDist)
2   double dist = 0
3   for int i = 1 to len(query)
4     dist += (sample[i] - query[i])^2
5     // early abandoning
6     if (dist > bestDist) return Double.MAX_VALUE
7   return dist
```

4.5 Search Space Pruning

The rationale of search space pruning is to early abandon unpromising candidates, if these will not result in finding a new optimum. Previous work aims at stopping Euclidean distance calculations when the current distance exceeds the best distance found so far [52, 63, 91].

4.5.1 Early Abandoning

The purpose of the *ShotgunDistance*-algorithm is to add up the Euclidean distances corresponding to each query window. We can stop the calculation of the Shotgun distance when the current sum exceeds the minimal known distance:

- Algorithm 4.5 illustrates the early abandoning algorithm used to abandon Euclidean distance computations [52, 63, 91]. The basic rationale is to stop the *for*-loop when the current sum exceeds the threshold (line 6) .
- The *ShotgunDistance*-algorithm (Algorithm 4.6) is executed multiple times for each pair of query and sample. This allows for two optimizations:
  1. Window pruning: By recording the minimal distance between a query window and a sample window, this distance can be passed to the Euclidean distance for pruning unpromising windows (line 7).
  2. Sample pruning: By recording the minimal distance between the query and the current 1-NN, we can prune unpromising sample time series when this threshold is exceeded (line 10).

Parallel Execution (Algorithm 4.6): The early abandoning techniques make the parallel execution of the Shotgun distance more complex. We have to make the distance threshold D (line 16) available to all processes, to allow for efficient early abandoning. While this can easily be achieved in a shared memory system, the problem becomes more complicated in a distributed environment like MapReduce, as the threshold D has to be propagated to all servers for each update.

Algorithm 4.6 Pruning techniques based on early abandoning.

```
1  double ShotgunDistance(TimeSeries query,TimeSeries sample,int w,bool mean_norm,double
       bestDist)
2    double totalDist = 0
3    for TimeSeries q in disjoint_windows(query, w, mean_norm)
4      double qDist = Double.MAX_VALUE
5      for TimeSeries s in sliding_windows(sample, w, mean_norm)
6        // early abandoning
7        qDist = min(qDist,D_ED(q,s,min(qDist,bestDist)))
8      totalDist += qDist
9      // early abandoning
10     if (totalDist > bestDist) return Double.MAX_VALUE
11   return totalDist
12
13 String predict(TimeSeries q, TimeSeries[] samples, int w, bool mean_norm)
14 [...]
15   for TimeSeries sample in samples // in parallel
16     D = min(D,ShotgunDistance(q, sample, w, mean_norm, D))
17 [...]
```

Computational Complexity: All these optimizations start from the assumptions that we compute the distances for a set of N samples of length n.

The early abandoning of the Euclidean distance reduces the computational complexity from $O(Nn)$ to a best case computational complexity of:

$$T(Euclid) \quad \epsilon \quad \Omega(1 \cdot n + (N-1) \cdot 1) \tag{4.12}$$

$$= \quad \Omega(n + N) \tag{4.13}$$

In the best case scenario, we have to compute one Euclidean distance with complexity $O(n)$ and all other $N-1$ distance computations can be stopped after one iteration.

As for the classification using the Shotgun distance the computational complexity can be reduced from $O\left(N \cdot (n^2 - nw)\right)$ to a best case computational complexity of:

$$T(Predict) \quad \epsilon \quad \Omega\left(1 \cdot (n^2 - nw) + (N-1) \cdot w\right) \tag{4.14}$$

$$= \quad \Omega\left(n^2 - nw + Nw\right) \tag{4.15}$$

In the best case scenario, we have to compute the whole Shotgun distance for the first time series with complexity $O(n^2 - nw)$ and can early abandon all other computations after calculating the Euclidean distance for the first query window.

4.5.2 Upper Bound on the Accuracy

While lower bounding on distance computations aims at reducing the complexity in the time series length n, we present a novel optimization that addresses the number of samples N. For each window length, the best achievable accuracy in Algorithm 4.7 (line 10) at any iteration is given by:

$$\text{correct} \leq (\text{current correct} + \text{remaining samples}) \leq N \tag{4.16}$$

Algorithm 4.7 Use an upper bound on the current accuracy.

```
1  [(int,int)] fit(TimeSeries[] samples, bool mean_norm)
2    [(int,int)] scores = []
3    int bestCorrect = 0
4    for int len = maxLen down to minLen
5      int correct = 0
6      for int q in [1..len(samples)]
7        String nnLabel = predict(samples[q],samples\{samples[q]}, len, mean_norm)
8        if (nnLabel==samples[q].label) correct++
9        int remaining = (len(samples)-q)
10       if (correct+remaining) < bestCorrect*factor
11         break
12     bestCorrect = max(bestCorrect,correct)
13 [...]
```

Thus, we do not need to obtain the exact accuracy for each window length (line 10), if the remaining samples will not result in finding a higher accuracy (or within *factor* to the best accuracy). This optimization is used for training the classifiers, as we might only have to test a few window lengths to find the best ones. That means, if we know that one window length scores a perfect 100%, we can stop iterating other window lengths after the first mislabeled sample.

Computational Complexity: The upper bound on the accuracy reduces the computational complexity when training the classifier from $O\left(\sum_{w=minLen}^{n} N \cdot T\left(\text{Predict}\right)\right)$ to a best case complexity of:

$$T(Fit) \quad \epsilon \quad \Omega\left(1 \cdot N \cdot T\left(\text{Predict}\right) + (n-1) \cdot T\left(\text{Predict}\right)\right) \tag{4.17}$$

In the best case scenario, we have to compute the cross-validation for the first window lengths with the complexity $O(N \cdot T\left(\text{Predict}\right))$ and can early abandon all other computations after the first mislabeled sample.

4.6 Experiments

4.6.1 Setup

We evaluate the Shotgun distance based on case studies for not preprocessed datasets and the established UCR benchmark datasets [42] (see Appendix Table 8.1). The webpage accompanying the Shotgun publication contains the raw numbers and the C++ source codes [70]. The Shotgun ensemble classifier was implemented in C++ using OpenMP and JAVA. All experiments were performed using the JAVA implementation on a shared memory machine running LINUX with 8 Quad Core AMD Opteron 8358 SE processors, and JAVA JDK x64 1.8. For all experiments the time series datasets were z-normalized prior to the experiments (compare Chapter 2.2). All experiments consist of two phases:

model building using the train dataset and testing the classification accuracy using the test dataset.

Each dataset provides a *train/test* split. There might be a confusion with the terms "training set", "validation set", and "test set" used in supervised learning. The train split is used as the training and validation set. The test split is used as the test set. By the use of these train/test splits, the results are comparable to previous publications. All our results are based on the *test accuracy* of the classifiers using the test split.

4.6.2 Case Studies: Hierarchical Clustering

Several kinds of invariances for distance measures were presented (compare Chapter 2.2). The Shotgun distance accounts for the following invariances:

1. Amplitude/offset, due to normalization,
2. Local scaling, due to breaking the query into disjoint windows,
3. Phase shifts, due to shifting the query windows along the sample, and
4. Occlusion in the samples, due to the use of subsequence to subsequence matching (extraneous data in the sample will be ignored).

This is underlined by the following case studies. Unfortunately, good clustering results do not imply good classification results and vice versa, as the symmetry condition is not satisfied by the Shotgun distance. A dendrogram plot (binary tree) is used to illustrate the results of a hierarchical clustering. The dendrogram makes use of u-shapes that connect the two most similar time series. The height of each u-shape is equal to the distance (similarity). We use the Shotgun distance for clustering as opposed to the Shotgun ensemble used for classification.

Hierarchical Clustering of Walking Motions: Figure 4.6 shows a dendrogram of a hierarchical clustering of walking motions of 4 subjects [26]. Each motion was categorized by the labels *normal* walk (green) and *abnormal* walk (orange). The difficulties in this dataset arise from variable length gait cycles, gait styles and paces due to different subjects throughout different activities that include stops and turns. A normal walking motion consists of multiple repeated similar gait cycles. The dataset requires amplitude, local scaling, phase, and occlusion invariances for a meaningful clustering. We set the two parameters of the Shotgun distance using cross-validation on the train dataset.

The ED does not provide phase invariance and fails to identify all abnormal walking styles. As such these cannot be separated from the normal walking motions. DTW provides some invariance to phase shifts (peak-to-peak and valley-to-valley alignment). This still does not result in a satisfying clustering as two walking motions with a different number of gait cycles (valleys and peaks) are likely to be misaligned. Our Shotgun distance

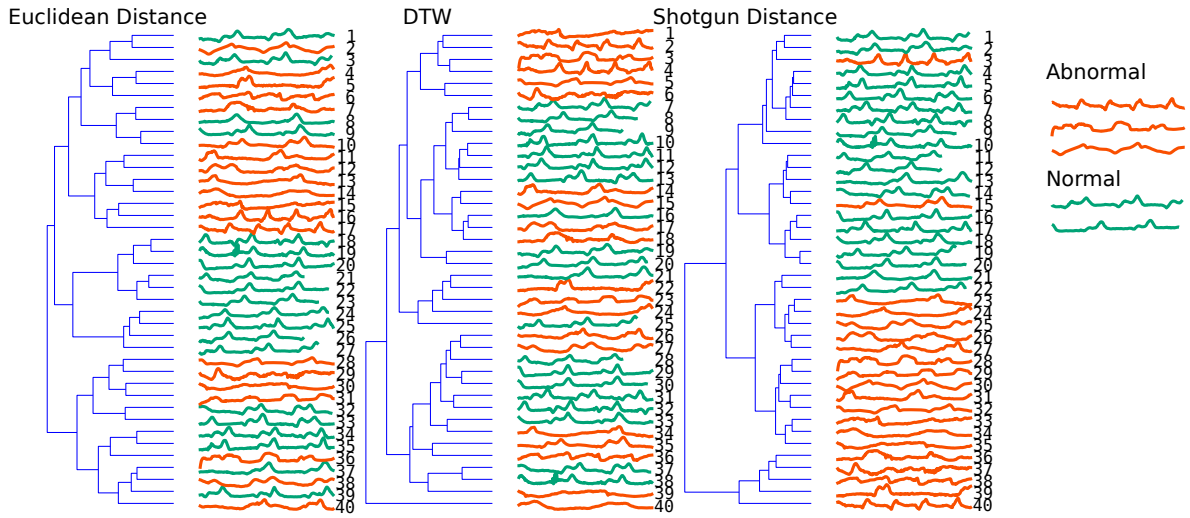


Figure 4.6 – Dendrogram of hierarchical clustering of walking motions. There are two types of walking motions: Normal (green) and Abnormal (orange) walk.

separates the normal walking motions from the abnormal walking motions clearly with just the 3rd and 15th abnormal walking motion being out of place.

Hierarchical Clustering of Heraldic Shields: Figure 4.7 shows a dendrogram of a hierarchical clustering of the shape of heraldic shields [90]. These shields come from three countries: Spanish (blue), Polish (orange), and French (green). The shapes of the shields differ based on their origin. The method used to transform the outline of a shield to a time series is described in [90]. Other than the previous case study, this dataset does not show any periodicity. For a meaningful clustering the dataset requires local and global scaling invariances. We set the parameters of the two Shotgun distance using cross-validation on the train dataset.

The ED clusters the time series based on their length and not on their shape, which results in a visually unpleasant clustering. The DTW perfectly clusters the Spanish shields but fails to distinguish the Polish from the French shields. Our Shotgun distance is the only similarity measure to separate all shields correctly.

Implications: The Shotgun distance provides invariances to amplitude/offset, local scaling, phase shifts, and occlusion. As such it provides a better clustering than the ED and DTW on two use cases for not pre-processed time series datasets.

4.6.3 Case Studies: Classification

Please refer to Chapter 4.6.1 regarding the use of the *train/test* splits for testing and training. We use the Shotgun ensemble classifier here.

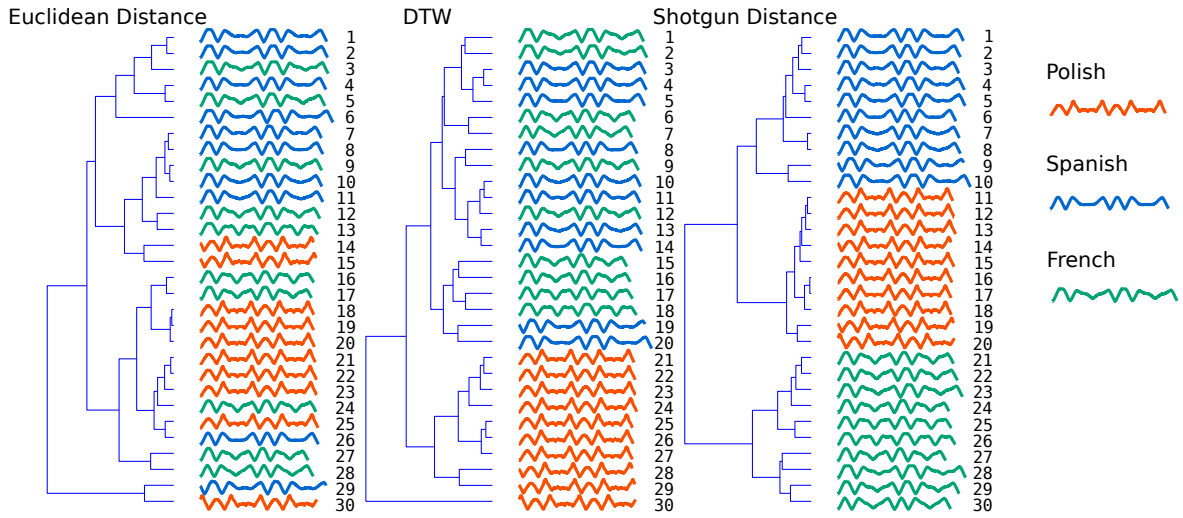


Figure 4.7 – Dendrogram of hierarchical clustering of heraldic shield outlines. There are three types of shields: Spanish (blue), Polish (orange) and French (green).

Dataset	1-NN DTW	State of the Art	Shotgun Ensemble Classifier	
Personalized Medicine	62.8%	92.4% [38]	99.5%	*factor* : 0.95, mean:true
Human Walking Motions	66.2%	91% [91]	96.9%	*factor* : 0.95, mean:true
Wheat Spectrographs	71.3%	72.6% [91]	80.7%	*factor* : 0.95, mean:true
Bio Acoustics	-	93.29% [1]	94.0%	*factor* : 1.00, mean:true
Astronomy	90.7%	93.68% [63]	95.3%	*factor* : 0.97, mean:true

Table 4.1 – Test accuracies on the case studies.

Personalized Medicine: The BIDMC Congestive Heart Failure Database [3] contains ECG recordings of 15 subjects who suffer from severe congestive heart failure (Figure 4.8 top left). The recordings contain noisy or extraneous data when the recordings started before the machine was connected to the patient. ECG signals show a high level of redundancy due to repetitive heart beats but even a single patient can have multiple different heart beats. To deal with these distortions a classifier has to provide invariance to amplitudes, uniform scaling, phase shifts and occlusion. The total size of this dataset is equal to 9 million data points (10 hours sampled at at 250 Hz). We used the train/test split provided by [38] with 600 time series each.

To the best of our knowledge, the best rivaling approach reported a test accuracy of 92.4% [38] and 1-NN DTW scores 62.8%. The Shotgun ensemble classifier obtains a close to perfect test accuracy of 99.5%. This is a result of the design of the Shotgun distance: ECG signals are composed of recurring patterns, which are distorted by all kinds of noise. To obtain this score, training the Shotgun ensemble classifier took roughly two days using

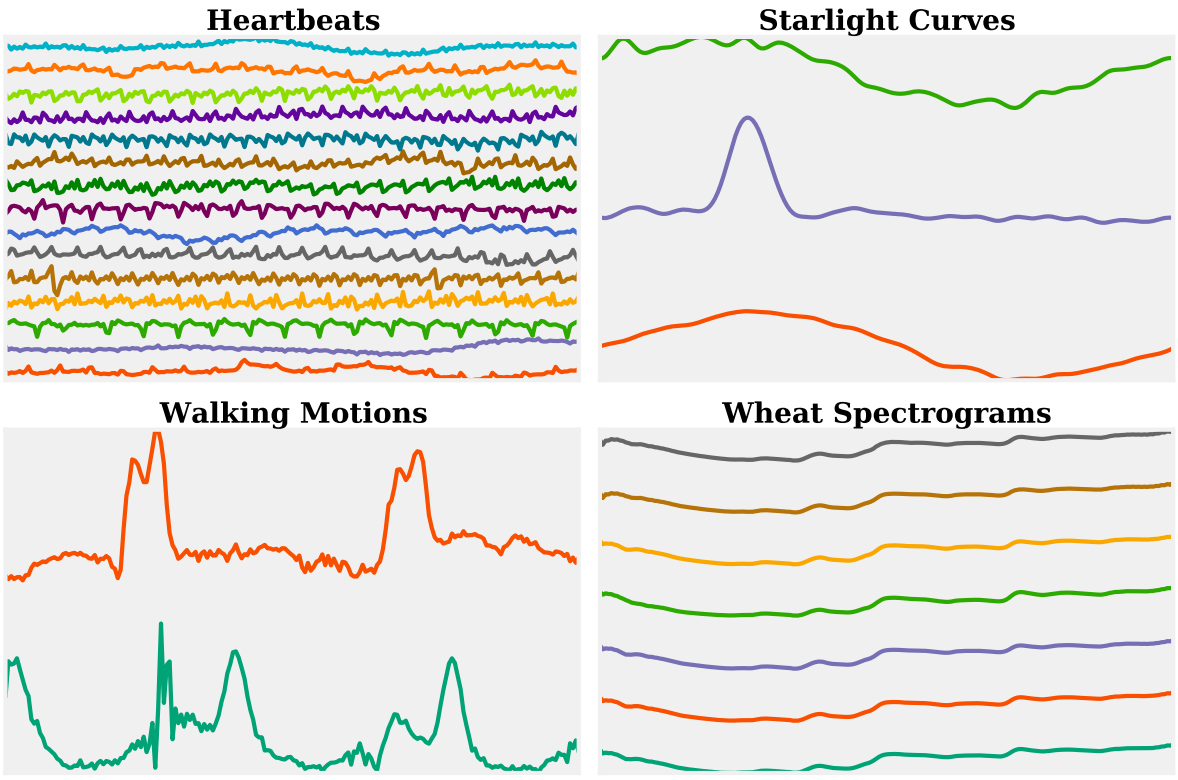


Figure 4.8 – A sample representing each class of the case studies: ECG signals, starlight curves, abnormal and normal walking motions, and wheat spectrographs.

all 32 cores as roughly 6000 window lengths had to be evaluated. Prediction on the 600 test samples took roughly 1.5 hours in total.

Our 1-NN BOSS ensemble classifier (Chapter 5), which is the successor of the Shotgun ensemble classifier, scores a perfect 100%.

Human Walking Motions: We reuse the CMU [26] walking motions of four subjects (Figure 4.8 bottom left). To make our results comparable to [91], we use the data provided by their first segmentation approach. We classify the normal and abnormal walking patterns.

Training the Shotgun ensemble classifier took less than one minute using all 32 cores. This results in a test classification accuracy of 96.9% due to the repetitive nature of the data. The accuracy is significantly higher than that of the best rivaling approach in [91] with an accuracy of 91% or 1-NN DTW that scores 66.2%.

Our 1-NN BOSS ensemble classifier (Chapter 5), which is the successor of the Shotgun ensemble classifier, scores a similar 97.4%.

Spectrographs: *Wheat* [91] is a dataset of 775 spectrographs of wheat samples grown in Canada. The dataset contains different wheat types like *Canada Western Red Spring, Soft White Spring* or *Canada Western Red Winter* (Figure 4.8 bottom right). The class labels define the year in which the wheat was grown. This makes the classification problem much more difficult, as the same wheat types in different years belong to different classes.

The best rivaling approach [91] reports a test accuracy of 72.6% and 1-NN DTW obtains a test accuracy of 71.3%. Our Shotgun ensemble classifier obtains a much higher test accuracy of 80.69%.

Our 1-NN BOSS ensemble classifier (Chapter 5), which is the successor of the Shotgun ensemble classifier, scores a similar 80.4%.

Implications: The Shotgun ensemble classifier is significantly more accurate than state of the art on case studies for not pre-processed time series datasets, with the exception of our 1-NN BOSS ensemble classifier (Chapter 5) that builds upon the alignment-free property of the Shotgun ensemble classifier and introduces tolerance to noise.

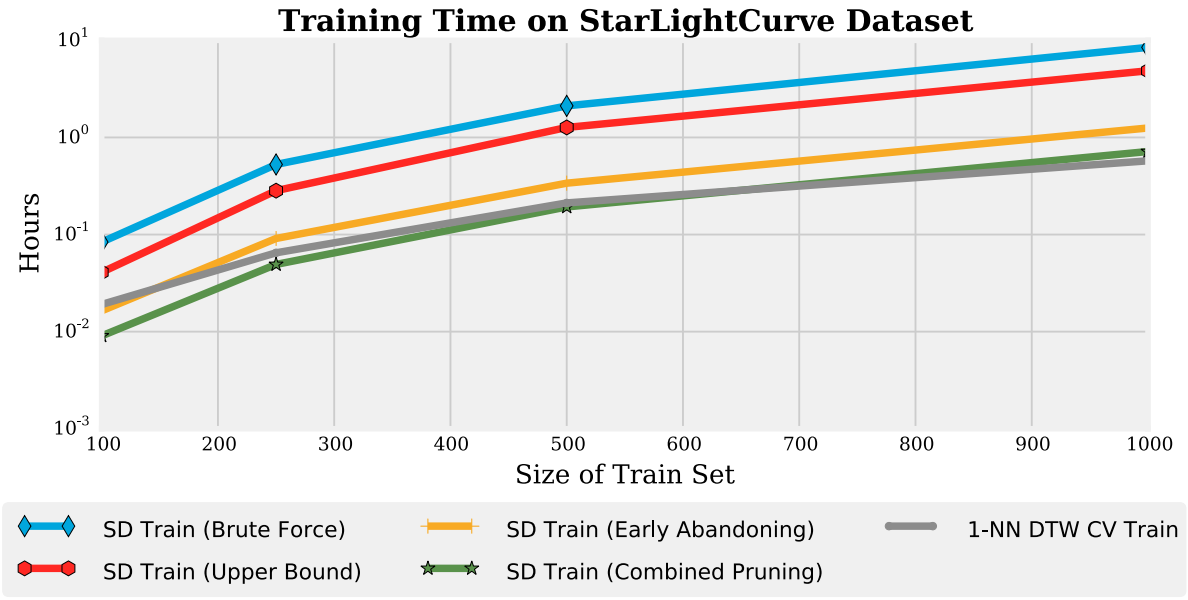


Figure 4.9 – The time required to execute the Shotgun *fit*-method on the *StarLightCurves* dataset using the presented pruning strategies.

Astronomy / Scalability: While it is easy to get large amounts of data, it can be very time consuming to obtain labels for each data item. Thus, it is difficult to obtain datasets with large amounts of labeled data. We test the utility of our pruning methods (Chapter 4.5) using the largest dataset *StarLightCurves* available in the UCR time series archive [42]. This dataset contains three types of star objects: *Eclipsed Binaries*, *Cepheids*

and *RR Lyrae Variables*. The *Cepheids (top)* and *RR Lyrae Variables (bottom)* have a similar shape and are hard to separate (Figure 4.8).

To the best of our knowledge, the highest reported test accuracy is 93.68% [63], and the 1-NN DTW scores 90.7%. The test accuracy of our Shotgun ensemble classifier is 95.3%.

Our 1-NN BOSS ensemble classifier (Chapter 5) that is the successor of the Shotgun ensemble classifier scores a higher 97.9%.

Scalability: To test the scalability of the Shotgun *fit*-method, we iteratively doubled the number of samples from $N = 100$ to 1000, each of length $n = 1024$, and measured the pruning strategies presented in this thesis:

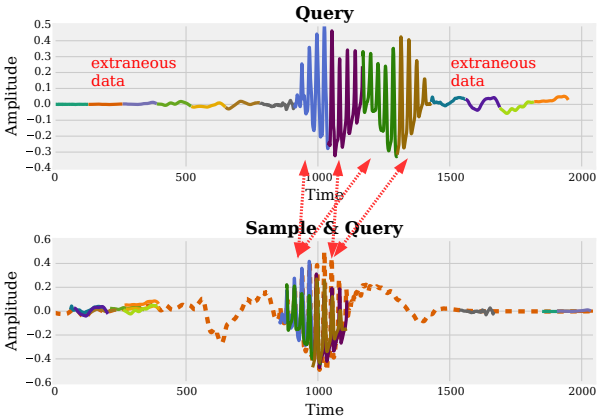
- Brute Force (Algorithm 4.3),
- Early abandoning Shotgun distance computations (Algorithm 4.6),
- Upper bound on accuracy (Algorithm 4.7),
- Combined upper bound and early abandoning (Algorithm 4.6 + Algorithm 4.7), and
- 1-NN DTW CV: with a warping window constraint set through cross-validation using the state of the art implementation [62] (compare Chapter 2.2).

All reported numbers were measured using 32 cores. Figure 4.9 shows that the train time of the *brute force* algorithm grows quadratically in N to approximately 9 hours for 1000 samples. *Early abandoning* reduces this by a factor of seven, and in combination with the *upper bound,* by a factor of 12 to only 41 minutes. The resulting train time is close to that of the 1-NN DTW CV classifier with a train complexity of $O(N^2n^2)$ using the state of the art implementation [62].

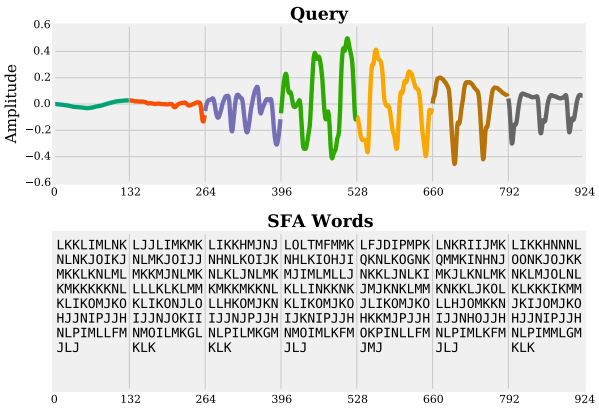
Implications: The presented pruning strategies significantly reduce the run-time for training by up to one order in magnitude. A more detailed runtime analysis will be presented in Chapter 6.

4.6.4 Case Study: Computational Bioacoustics

Producers set up traps in the field that lure and capture pests, in order to detect and count these. Manual inspection of traps is a procedure that is both costly and error prone. Repeated inspections must be carried out manually, sometimes in areas that are not easily accessible. A novel recording device has been recently introduced [1]. The core idea is to embed a device in an insect trap to record the fluctuations of light received by a photoreceptor as an insect passes a laser beam and partially occludes the light. The samples are recorded at 16 kHz and are 1s long but the actual insect motion within each



(a) An insect passes the laser twice, causing an *echo* (top). The Shotgun classifier extracts the characteristic patterns form the query and aligns the two signals (bottom).



(b) A closeup of an insect passage: the query is cut into windows (top), and an SFA word for each query window is calculated (bottom).

Figure 4.10 – Recordings of flying insects.

recording is typically only a few hundredths of a second long. The bandwidth between 0.2-4 kHz is most characteristic. We focus on a dataset [58] that was collected from five insects, namely:

1. *AEDES AEGYPTI male* (yellow fever mosquito),
2. *DROSOPHILA MELANOGASTER mixed sex (fruit flies)*,
3. *CULEX QUINQUEFASCIATUS female (southern house mosquito)*,
4. *CULEX TARSALIS female (mosquito)*, and
5. *CULEX TARSALIS male (mosquito).*

The dataset consists of a train/test split with 500 and 5000 recordings. Figure 4.10a shows two example insect recordings.

To connect time series analysis with bioacoustics, we use SFA (Chapter 3). In Chapter 3 we showed that its symbolic and thus compact representation of a time series allows for efficient similarity search and permits to index terabyte-sized datasets in main memory. Here we make use of the noise canceling property of the SFA transformation and the frequency domain.

Our workflow consists of feature extraction and feature matching. SFA is applied to extract features (SFA words), which are then passed to the Shotgun classifier. Algorithm 4.8 shows the change in concept. Instead of the Euclidean distance, we use the SFA distance (Equation 3.26) on two SFA words in line 10.

Algorithm 4.8 The SFA Shotgun distance.

```
double ShotgunDistance(TimeSeries query, TimeSeries sample, int w, bool mean_norm)
  double totalDist = 0.0
  // for each disjoint query window
  for TimeSeries q in disjoint_windows(query, w, mean_norm)
    qSFA = SFA(q)
    double qDist = Double.MAX_VALUE
    // find the position that minimizes the SFA distance
    for TimeSeries s in sliding_windows(sample, w, mean_norm)
      sSFA = SFA(s)
      qDist = min(qDist,D_SFA(qSFA,sSFA))
    totalDist += qDist
  return totalDist
```

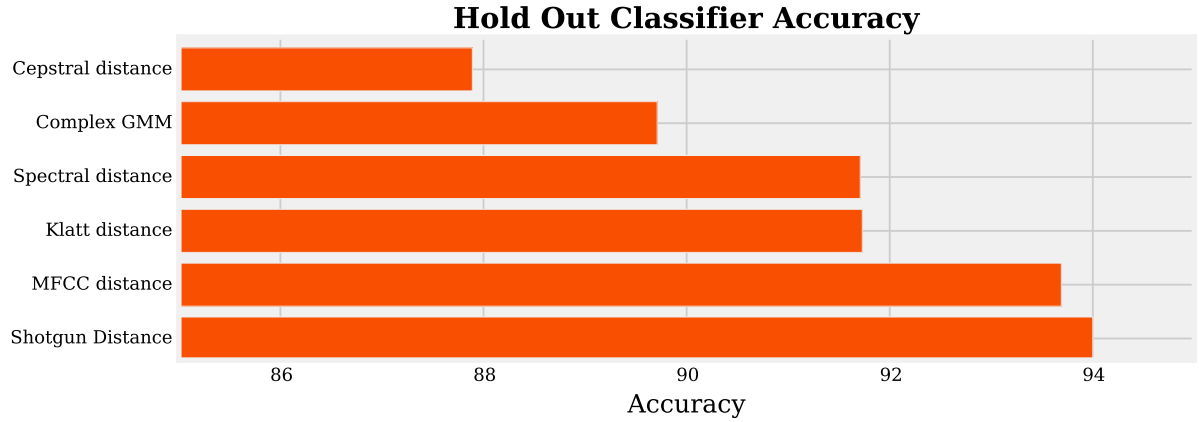


Figure 4.11 – Classification accuracies on the flying insects test dataset.

Noise is generated by the angle and speed of an insect passing the photoreceptor. This affects the recorded intensities. SFA reduces this noise by the use of low-pass filtering and quantization and thus accounts for these differences in the intensity. By introducing the Shotgun classifier for feature matching, we obtain invariance to phase shifts, i.e., the exact time point of the insect passage. Shotgun distance further deals with outliers like multiple insect passages within a short time frame (see Figure 4.10a top).

We compared the Shotgun distance to common feature extraction techniques in computational bioacoustics like [58]:

1. Mel Frequency Cepstral Coefficients (MFCC),
2. The Klatt spectrum,
3. The Spectral distance,
4. Complex Gaussian Mixture Models (GMM), and
5. The Cepstral Distance.

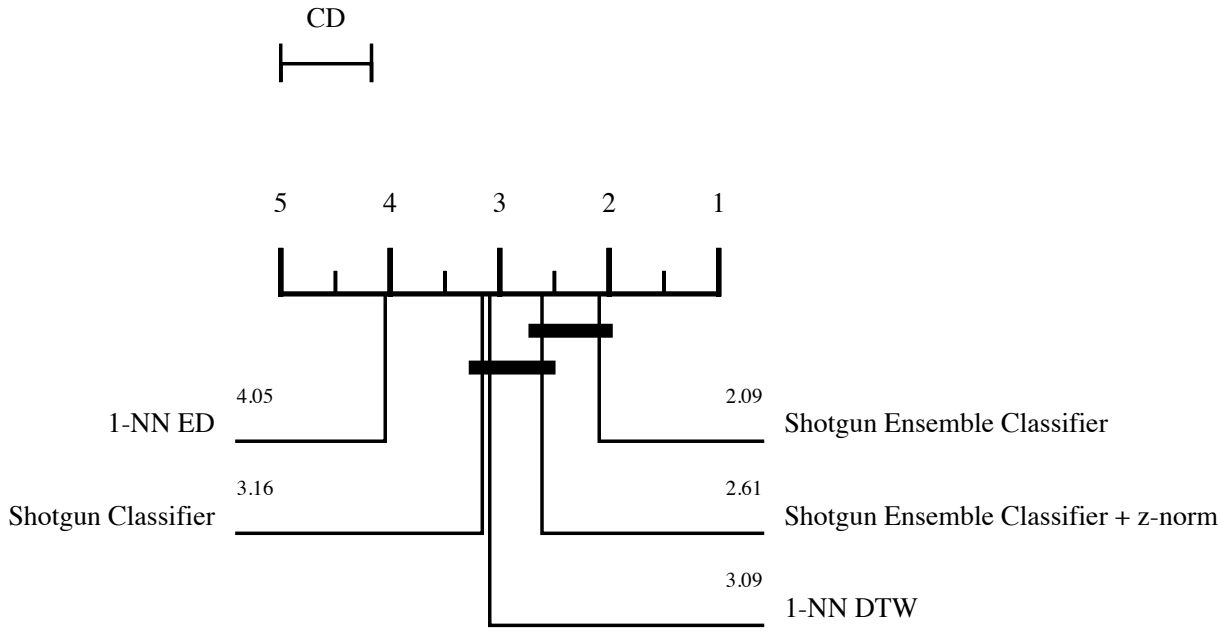


Figure 4.12 – Critical difference diagram for the different design decisions made using the 45 UCR datasets. The best classifiers are to the right. The critical difference (CD) is 0.83.

We used cross-validation on the train set to obtain the best parameters. For SFA using a small window length of $n = 132$, $l = 120$ SFA word length and $c = 22$ symbols performed best. Our approach scores the highest test accuracy with 94% [58] (Figure 4.11). The other feature extraction techniques give poorer performance.

Implications: This experiment shows that in combination with SFA the Shotgun distance can be used for computational bioacoustics and performs better than state of the art feature extraction techniques.

The combined SFA and Shotgun distance approach is the first step towards our BOSS model (Chapter 5) but differs in the use of the distance measure and the use of disjoint query windows. The BOSS model uses sliding query windows and a Euclidean like distance on the histogram of SFA words.

4.6.5 Impact of Design Decisions

We use all 46 UCR datasets [42] to test the impact of our design decisions (Table 8.1). The Shotgun ensemble classifier is based on two design decisions:

1. *Shotgun ensemble classifier*: Building an *ensemble* of Shotgun classifiers.

2. *Mean normalization*: Use the *mean normalization* as a parameter as opposed to always z-normalizing all windows prior to distance computations (*Shotgun Ensemble Classifier + z-norm*).

We decided to use 1-NN classification as it does not introduce any additional parameters for model training which allows us to focus solely on the parameters of the Shotgun ensemble classifier.

Figure 4.12 shows a critical difference diagram over the average ranks of the classifiers as introduced in [28]. The classifiers with the lowest (best) ranks are to the right. The group of classifiers that are not significantly different in their rankings are connected by a bar. The critical difference (CD) length is shown above the graph.

Overall the Shotgun ensemble classifier shows the best rank (Figure 4.12). Always performing z-normalization, reduces the rank by 0.52 points. The Shotgun classifier with a single window length performed worst.

Implications: The use of mean normalization as a parameter and an ensemble of Shotgun classifiers significantly improves classification accuracy, as time series datasets have characteristic patterns of variable lengths and the mean can be characteristic for the similarity of patterns.

4.6.6 Classification Accuracy Benchmark

For a comparison to the state of the art classifiers in terms of the classification accuracy and classification times please refer to Chapter 6.

4.7 Summary

Common similarity measures require the data to be pre-processed by domain experts for equivalent-length, and approximately aligned substructures (the alignment). Searching for similarities in substructures rather than matching the time series as a whole is the basic rationale of the Shotgun distance to deal with the *alignment-free* similarity of long time series. The Shotgun distance breaks a query time series into subsequences and vertically aligns and horizontally scales each subsequence prior to measuring the similarity to a sample time series, thereby reducing the cost-ineffective pre-processing for alignment. An ensemble classifier composed of 1-NN Shotgun classifiers is presented. To deal with the increased complexity, two pruning strategies are presented that reduce the computational complexity by one order of magnitude. In our experimental evaluation we show that the Shotgun distance performs better than rivaling methods in the context of hierarchical clustering and classification for use cases in computational bioacoustics, human motion detection, spectrographs, astronomy, or personalized medicine.

The Shotgun distance model offers high accuracy but has a high computational complexity. The Shotgun distance model first introduced (a) the ensemble of classifiers for different window lengths, (b) the use of the mean normalization as a parameter, (c) the idea of alignment-free similarity using subsequence to subsequence matching, and (d) the use of SFA for noise reduction applied to insect classification. These presented algorithms

are the foundation for the subsequent similarity models of this thesis (the BOSS model in Chapter 5 and the BOSS VS model in Chapter 6).

Chapter 5

Bag-Of-SFA-Symbols: Alignment-free Time Series Data Analytics in the Presence of Noise

The BOSS model is the second time series model presented in this thesis (compare Figure 1.2). This chapter gives a detailed description of the BOSS model. The chapter is based on and contains text passages from my publication [68].

5.1 Introduction

Raw time series data may be recorded at variable lengths, or are composed of repetitive substructures. These build a foundation for state of the art algorithms such as our Shotgun distance as presented in Chapter 4. However, extraneous or erroneous data, as a result of noise, have been paid surprisingly little attention to in literature. It has commonly been assumed that noise is filtered as part of a pre-processing step carried out by a human.

Figure 5.1 shows a dendrogram of a hierarchical clustering of the first 6 samples from the synthetic Cylinder-Bell-Funnel (CBF) dataset. This synthetic time series benchmark dataset is widely used and contains three basic shapes: Cylinders, Bells and Funnels. For the human eye the distinguishing power of the first two common distance measures is very disappointing. The Euclidean distance (ED) fails to cluster the funnel curves 1 and 6 as it does not provide horizontal alignment (phase invariance). The Dynamic Time Warping (DTW) distance provides local scaling (through its warping invariance), but still does not give a satisfying clustering as the funnel curves 4 and 5 are separated. Our Bag-Of-SFA-Symbols (BOSS) model clusters the funnel curves 1-2 and cylinder curves 3-5 correctly. This small example illustrates some desirable invariances for time series similarity as described in Chapter 2.2. The CBF dataset requires invariance to phase (horizontal alignment), warping (local scaling), occlusion, amplitude/offset, and noise.

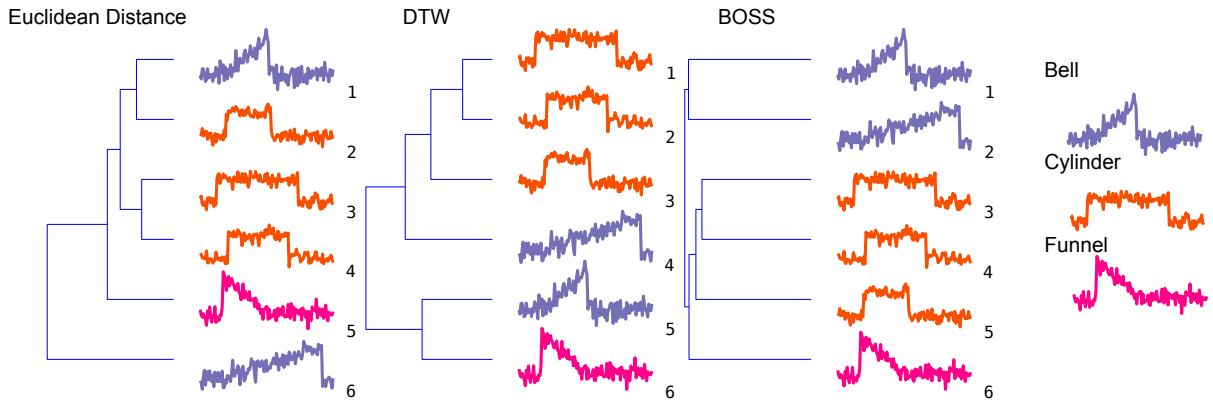


Figure 5.1 – Hierarchical clustering of the Cylinder-Bell-Funnel dataset based on three similarity metrics. There are three types of curves: Cylinder, Bell, and Funnel.

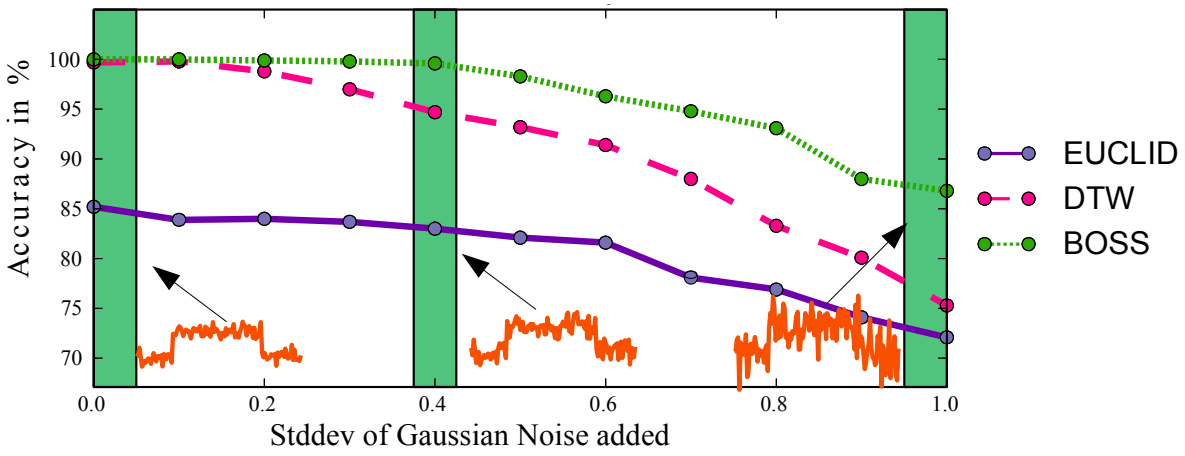


Figure 5.2 – Effects of Gaussian noise on classification accuracy when added to the Cylinder-Bell-Funnel dataset.

We believe that *invariance to noise* was paid too little attention to, as most time series data analytics algorithms are applied directly to the raw data. To illustrate the relevance of noise to the classification task, we performed another experiment on the CBF dataset. All time series were first z-normalized to a standard deviation (SD) of 1. We then added Gaussian noise with an increasing SD of 0 to 1.0 to each time series, equal to a noise level of 0% to 100%. Figure 5.2 shows that DTW and BOSS provide the best classification accuracies. With an increase of noise the classification accuracies decrease. The BOSS classifier is the most robust to noise and remains stable up to a noise level of 40%, whereas DTW degenerates from a noise level of 10% upwards.

Our BOSS model is a structure-based similarity measure that applies noise reduction to the raw time series. It first extracts substructures (patterns) from a time series. Next, it applies low-pass filtering and quantization to the substructures using the SFA representation. This reduces noise and allows for string matching algorithms to be applied.

Two time series are then compared based on the differences in the set of noise reduced patterns. As opposed to rivaling methods the BOSS model offers two advantages: (a) it applies noise reduction, and (b) it is a structure based similarity measure (alignment-free). As a result the BOSS model is the most accurate similarity measure we are aware of.

Our contributions are as follows:

- We present our BOSS model that combines the noise tolerance of the Symbolic Fourier Approximation (SFA) [76] with the structure-based representation of the bag-of-words model [48] (Chapter 5.3).
- We present optimization strategies of the BOSS model to reduce the computation complexity (Chapter 5.5).
- We present the *1-NN BOSS ensemble classifier* based on multiple BOSS models at different window lengths (Chapter 5.4).
- We show (Chapter 5.6) that the 1-NN BOSS ensemble classifier
  - achieves the highest accuracy when compared to any other rivaling classification model on case studies in diverse application areas,
  - shows the best classification accuracy on the UCR time series benchmark datasets up to this day.
  - provides better clusterings than ED and DTW on two case studies for raw time series datasets.

5.2 Background and Related Work

Existing classification algorithms can be categorized as (compare Chapter 4.2):

- *Shape-based:* These are based on the whole time series and perform a point-wise comparison. Examples include 1-NN Euclidean Distance (ED), 1-NN Longest Common Subsequence [87], or 1-NN DTW [65, 62] with a consensus that DTW is among the best time series similarity measures [29, 13, 49]. The problem with shape-based techniques is that these fail to classify noisy or long time series [38], such as the ones we consider in this chapter.
- *Structure-based:* These techniques extract higher-level feature vectors or build a model from the time series prior to make existing data mining algorithms applicable like 1-NN, SVMs, decision trees, or random forests [12, 38, 48, 52, 63, 91, 88]. Feature extraction techniques include the dimensionality reduction techniques (Chapter 2.3) or Shapelets [92, 52, 90, 91, 63]. We focus on these approaches in our experimental analysis.

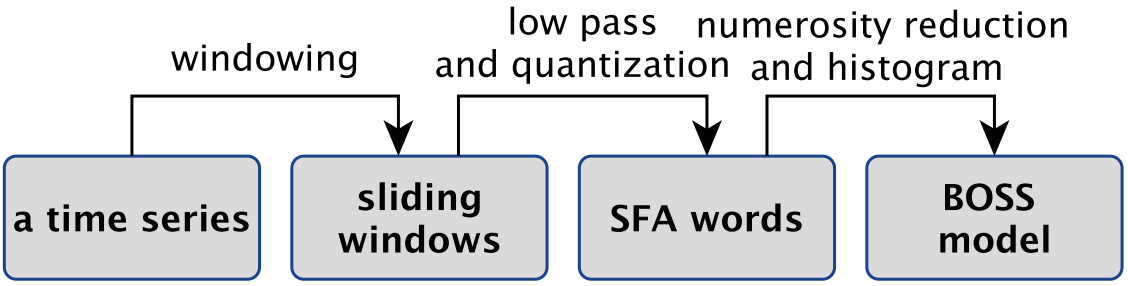


Figure 5.3 – The BOSS workflow: From a time series to the BOSS model.

The *bag-of-patterns* (BOP) model [48] is closest to the BOSS model. BOP is a structure-based technique as it extracts subsequences as higher-level features of a time series. BOP transforms these subsequences using the symbolic representation SAX, builds a histogram on the SAX words and uses the Euclidean Distance on these histograms as a similarity metric. Unlike SAX, which uses mean values (PAA) to approximate a time series, SFA uses DFT coefficients (frequency domain nature, compare Figure 3.1).

SAX-VSM [81] builds on BOP by the use of the *term frequency-inverse document frequency* (tf-idf) weighting of the histograms and the Cosine similarity as the similarity metric. It builds one histogram for each class, as opposed to one histogram for each sample. In contrast the BOSS model uses SFA [76], the offset invariance as a model parameter, a different similarity metric based on the Euclidean distance, and an ensemble of BOSS models. Time-series bitmaps [45] are a visualization tool for time series datasets based on a histogram of SAX words. The approach is similar to the BOP model.

The BOSS model is the successor of the Shotgun distance. The BOSS model makes use of several design decisions of the Shotgun distance model such as (a) the ensemble of classifiers, (b) offset invariance as a parameter, (c) and alignment-free similarity using subsequence to subsequence matching. It adds robustness to noise to the Shotgun distance model by a change of representation using SFA, and significantly reduces the computational complexity. In contrast, the Shotgun distance model is applied directly to the raw data, which makes it sensitive to noise.

5.3 The Bag-Of-SFA-Symbols: An Alignment-free and Noise-Robust Similarity Model

5.3.1 Motivation

Before going into the details of our Bag-Of-SFA-Symbols (BOSS) model, we present the building blocks in Figure 5.3. First, sliding windows of fixed length are extracted from

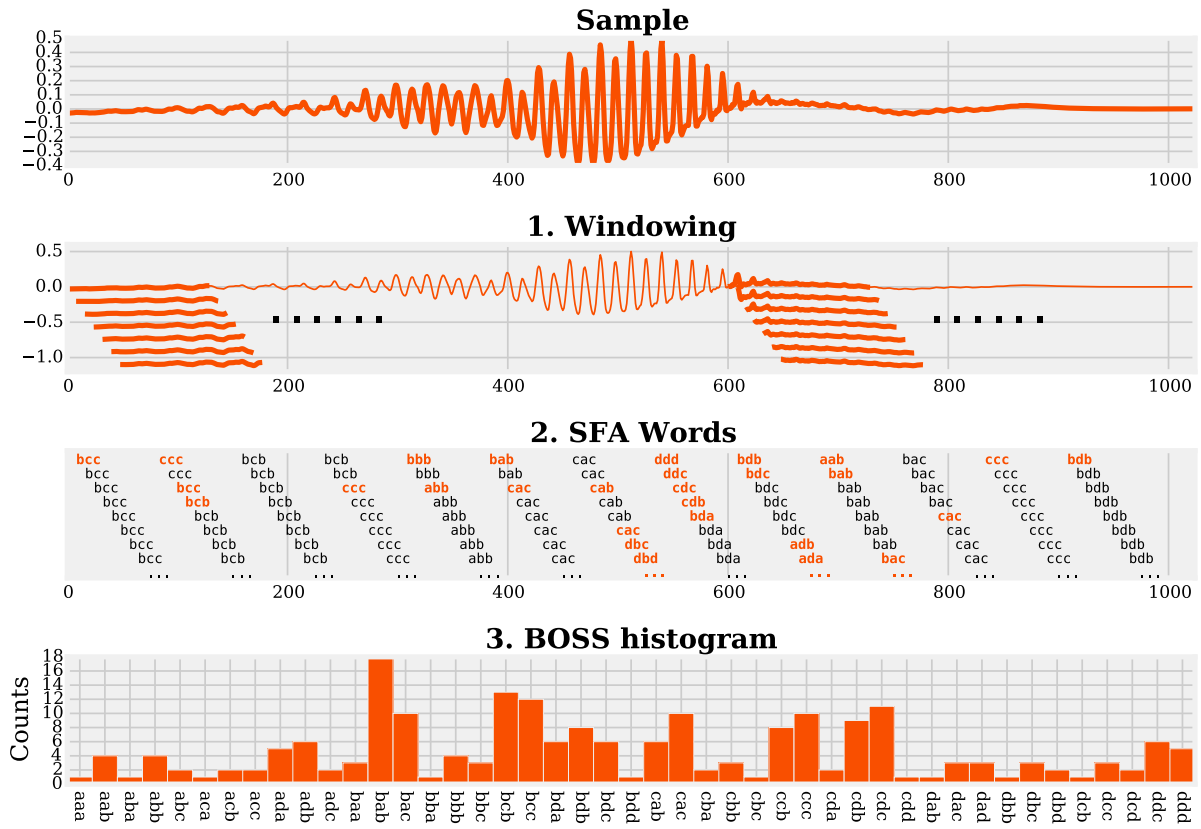


Figure 5.4 – The BOSS model is built from a sample time series using SFA word length 3 and 4 symbols (a-d). The black SFA words are skipped due to the numerosity reduction step and the red SFA words are recorded in the histogram (bottom).

a time series. Next, the symbolic representation *Symbolic Fourier Approximation* (SFA) is applied to each sliding window. SFA provides low-pass filtering and quantization to reduce noise (Chapter 3.3). This results in one SFA word for each sliding window. The histogram over the SFA words (BOSS model) can then be used as the indicator for the structural similarity of time series. Figure 5.4 illustrates the BOSS model for a sample time series using an SFA word length of 3 and 4 symbols.

The BOSS model describes each time series as an unordered set of substructures using SFA words. This approach has multiple advantages:

1. It is *alignment-free* as it extracts substructures and compares two time series based on their structural similarity, given by the SFA word frequencies,

2. It provides invariances to phase shifts, local scaling, offsets, amplitudes, and occlusions,

3. It applies *noise reduction* by the use of SFA, and

4. It is faster than the Shotgun distance, as hashing is used to determine the similarity of SFA words.

Properties (1) to (2) are provided by the Shotgun distance model. However, the Shotgun distance model directly compares time series based on the raw data. The BOSS model additionally provides properties (3) to (4) by the change of representation using SFA.

The BOSS webpage [72] contains a video illustrating the transformation of a time series to the BOSS model.

5.3.2 The BOSS Model

Our BOSS model has four parameters (Figure 5.4):

- **The window length** $w \epsilon \mathbb{N}$: Represents the size of the substructures.
- **Mean normalization** $mean \epsilon [true, false]$: Set to true for offset invariance.
- The **SFA word length** $l \epsilon \mathbb{N}$ **and alphabet size** $c \epsilon \mathbb{N}$: Used for low-pass filtering and the string representation.

First, sliding windows of length w are extracted from a time series. Intuitively w should roughly represent the size of the substructures within the time series. Next, each sliding window is normalized to have a standard deviation of 1 to obtain amplitude invariance. The parameter *mean* determines if the mean value is to be subtracted from each sliding window to obtain offset invariance. The mean normalization is treated as a parameter of the BOSS model and can be enabled or disabled. For example, heart beats should be compared using a common baseline (mean=true) but the pitch of a bird sound can be significant for the species (mean=false). Finally, the SFA transformation is applied to each real-valued sliding window.

The BOSS model transforms a time series into an unordered set of SFA words. Using an unordered set provides invariance to the horizontal alignment of the substructure contained in the time series (phase shift + local scaling invariances). It thereby eliminates the need for pre-processing the samples by a domain expert for approximate alignment of the substructures.

Numerosity Reduction [48, 47]: In stable sections of a signal, the SFA words of two neighboring sliding windows are very likely to be identical. To avoid outweighing stable sections of a signal, *numerosity reduction* is commonly applied. That means, the first occurrence of an SFA word is counted and all duplicates are ignored until a new SFA word is detected. In Figure 5.4 the first SFA words are identical:

$$S = \textbf{bcc}\ bcc\ bcc\ bcc\ bcc\ bcc\ bcc\ bcc\ \textbf{ccc}\ ccc\ \textbf{bcc}\ \textbf{bcb}\ bcb\ bcb\ bcb\ ... \quad (5.1)$$

Algorithm 5.1 The BOSS transformation.

```
1 map<String,int> BOSSTransform(TimeSeries sample,int w,int l,int c,bool mean)
2   map<String,int> bossHist = []
3   for TimeSeries S in sliding_windows(sample,w) // in parallel
4     String word = SFA(S,l,c,mean)
5     if word != lastWord      // numerosity reduction
6       bossHist[word]++       // increase histogram counts
7     lastWord = word
8   return bossHist
```

Applying numerosity reduction to S this leads to:

$$S' = bcc\ ccc\ bcc\ bcb\ ... \tag{5.2}$$

From these SFA words a histogram is built, which counts the occurrences of the SFA words. In the above example the BOSS histogram of S' is:

$$B:\ bcc\ =\ 2,\ \ ccc = 1,\ \ bcb = 1, ... \tag{5.3}$$

This BOSS histogram ignores the ordering of the SFA word occurrences within a time series.

Definition 13. Bag-Of-SFA-Symbols (BOSS): Given are a time series T, its sliding windows $S \,\epsilon\, windows(T, w, 1)$ (see Equation 2.5) and SFA transformations

$$SFAs(T) = \{SFA(S)\,|S \,\epsilon\, windows(T, w, 1)\}$$

The BOSS histogram (BOSS model) $B:\ \Sigma^l \rightarrow \mathbb{N}$ is a function of the SFA word space Σ^l to the natural numbers. The number represents the occurrences of an SFA word within $SFAs(T)$ counted after numerosity reduction.

5.3.3 The BOSS Transformation

The BOSS transformation (Algorithm 5.1) extracts sliding windows of length w from the sample (line 3) and calculates SFA words (line 4) with SFA word length l and alphabet size c. Mean normalization is obtained by dropping the first Fourier coefficient in each SFA word. Finally, the new SFA word is added to the BOSS histogram (lines 5–6), if two subsequent SFA words are different (numerosity reduction).

Parallel Execution (Algorithm 5.1): Calculating the BOSS transformation is composed of independent parts: Each sliding window can be simultaneously transformed to its SFA word and the update of the histogram has to be synchronized.

Figure 5.5 – The BOSS model of normal (green) and abnormal (orange) walking motions.

Computational Complexity: The BOSS model has a computational complexity linear in n: there are $n-w+1$ sliding windows of length w in a time series of length n. The first sliding window has to be transformed using the DFT with a complexity of $O(w \log w)$. The remaining $(n-w)$ sliding windows can be transformed using the MFT with a complexity of $O(l)$, for SFA word length l. As l is fixed to 16, l can be considered constant in the window length w (compare Chapter 3.3.7):

$$
\begin{aligned}
T(BOSS) \quad &\epsilon \quad O(1 \cdot T(SFA_{whole}) + (n-1) \cdot T(SFA_{subsequence})) && (5.4)\\
&\epsilon \quad O(w \log w + (n-w) \cdot l) \qquad \text{, for } l \leq 16 && (5.5)\\
&= \quad O(n + w \log w) && (5.6)
\end{aligned}
$$

5.3.4 The BOSS Distance

Two time series are considered similar if they share the same set of SFA words. Figure 5.5 illustrates the BOSS histograms for abnormal (orange) and normal walking motions (green). The walking motions contain noise, erroneous data (a peek in the first motion), a variable number of gait cycles, and the gait cycles are not horizontally aligned. Still the BOSS histograms for the first and second normal walking motion are very similar, while the histograms of the abnormal motions 3 and 4 clearly differ from the first two.

When comparing two BOSS histograms, the absence of SFA words can have two reasons:

1. Noise distorts the window leading to different SFA words, or
2. A substructure/window is not contained in the other time series.

Let us assume two sensors started recording the ECG of a patient, but one of the sensors was connected to a patient later in time. This sensor will contain extraneous data at the beginning when it was not connected. The two resulting time series will have identical BOSS histograms except for the SFA words at the beginning of the recording. To deal with this, the BOSS histogram has to deliberately ignore some SFA words for the signals to become identical. Thus, we chose to add tolerance to missing SFA words in our distance measure. The BOSS distance is a modification of the Euclidean distance: we omit all SFA word counts of 0 *in the query* when computing the pairwise differences. For example, the BOSS histograms of the first and fourth motion in Figure 5.5 are:

	aaa	*aab*	*aba*	*abb*	*baa*	*bab*	*bba*	*bbb*
$B_1 =$	3	4	4	4	0	0	1	3
$B_4 =$	1	0	1	0	3	5	3	4

The resulting BOSS distances are $D_{BOSS}(B_1, B_4) = 4 + 16 + 9 + 16 + 4 + 1 = 50$ and $D_{BOSS}(B_4, B_1) = 4 + 9 + 9 + 25 + 4 + 1 = 52$:

	aaa	*aab*	*aba*	*abb*	*baa*	*bab*	*bba*	*bbb*
$D_{BOSS}(B_1, B_4) =$	$(3-1)^2$	$+(4)^2$	$+(4-1)^2$	$+(4)^2$	$+\mathbf{0}$	$+\mathbf{0}$	$+(1-3)^2$	$+(3-4)^2$
$D_{BOSS}(B_4, B_1) =$	$(3-1)^2$	$+\mathbf{0}$	$+(4-1)^2$	$+\mathbf{0}$	$+(3)^2$	$+(5)^2$	$+(3-1)^2$	$+(3-4)^2$

Definition 14. BOSS distance: Given two BOSS histograms $B_1 : \Sigma^l \to \mathbb{N}$ and $B_2 : \Sigma^l \to \mathbb{N}$ of two time series T_1 and T_2, the BOSS distance is defined as:

$$D(T_1, T_2) = dist(B_1, B_2) \tag{5.7}$$

with

$$dist(B_1, B_2) = \sum_{t \epsilon B_1;\, B_1(t) > 0} [B_1(t) - B_2(t)]^2 \tag{5.8}$$

This BOSS distance is not a distance metric as it neither satisfies the symmetry condition nor the triangle inequality (compare Chapter 2.2). As a consequence the BOSS distance does not allow for exact indexing (triangle inequality) and if C is the nearest neighbor of a time series Q, Q does not have to be the nearest neighbor of time series C (symmetry condition).

In the context of time series data analytics this BOSS distance gave the best classification accuracy. However, other distance metrics such as the Euclidean distance or Cosine similarity may be applied, if the two above mentioned conditions have to be met. We compare the influence of these two distance metrics on the classification accuracy in Chapter 5.6.3.

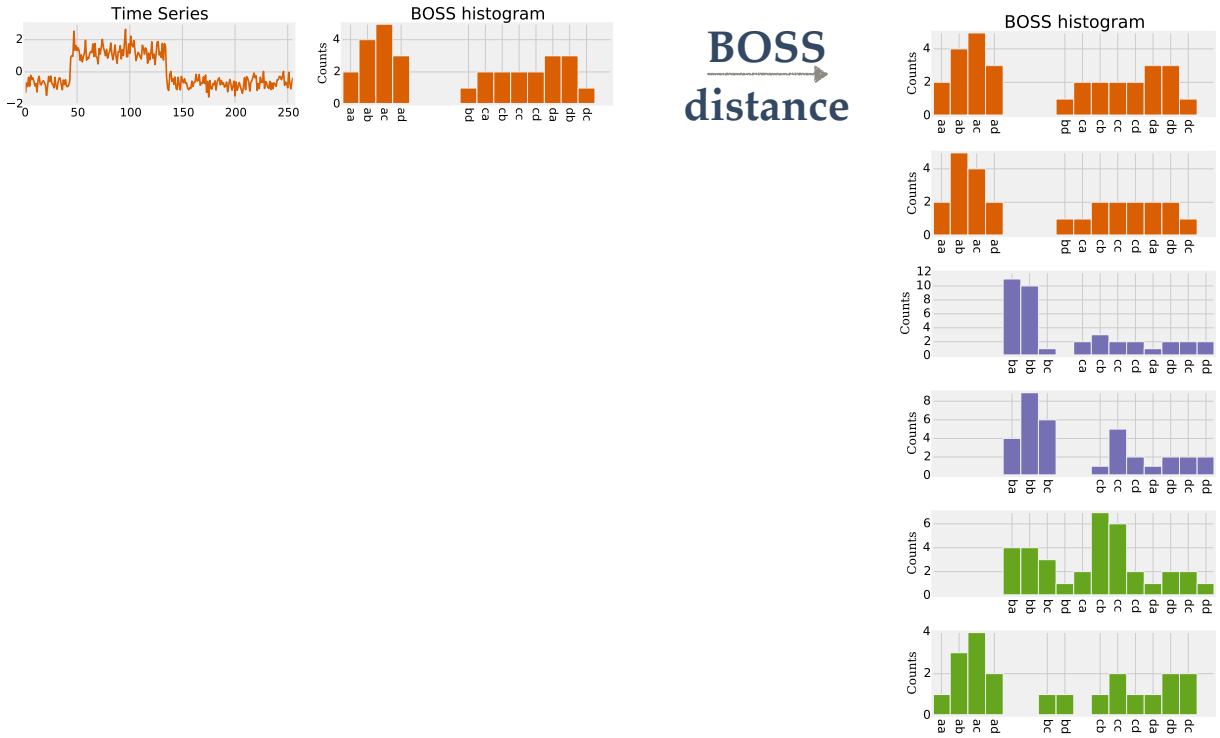


Figure 5.6 – 1-NN classification using the BOSS model: it labels a query by the label of the 1-NN to the query.

Computational Complexity: The computational complexity of the BOSS distance is linear in the length of the time series n. Each BOSS histogram contains at most $n-w+1$ unique SFA words. A BOSS histogram lookup for an SFA word has a constant computational complexity of $O(1)$ by the use of hashing. This results in a total computational complexity of:

$$T(BOSSDistance) \quad \epsilon \quad O((n-w+1)\cdot 1) \tag{5.9}$$
$$= \quad O(n) \tag{5.10}$$

While the computational complexity is upper bound by the time series length n, the actual number of *unique* SFA words is typically much smaller due to the numerosity reduction step and the use of histograms.

5.4 Time Series Classification

Classification describes the task of assigning a label to an unlabeled time series Q using a trained model. Figure 5.6 illustrates this classification task. The 1-NN BOSS classifier is a 1-NN classifier. That means, it uses a set of train samples DS as the model and

Algorithm 5.2 Predict: 1-NN classification using the BOSS model.

```
1  String predict(map<String, int> qHist, map<String,int>[] sHist)
2    (double bestDist, int nn) = (Double.MAX_VALUE, -1)
3    for int i in [1..len(sHist)]
4      double D = 0
5      // iterate only those words with a count > 0!
6      for (word,count) in qHist
7        D += (count-sHist[i].get(word))^2
8      if D < bestDist // store current 1-NN
9        bestDist = D
10       nn = i
11   return label(i)
```

performs a 1-NN search to find the label of Q. Thus, the query is assigned to the class of the time series T that minimizes the BOSS distance:

$$label(Q) = \underset{T \epsilon DS}{argmin}(D_{BOSS}(BOSS(Q), BOSS(T)))$$

The four parameters of the BOSS model are part of the classification model. As the labels of the train time series have to be known beforehand, this is referred to as supervised learning.

5.4.1 The BOSS Classifier Algorithm

Our BOSS classifier is based on 1-NN classification and the BOSS model. We decided to use 1-NN classification as it is very robust and does not introduce any additional parameters for model training [29].

Prediction (Algorithm 5.2): Given a query, the *predict*-method in Algorithm 5.2 searches for the 1-NN to a query within a set of samples by minimizing the BOSS distance between the query histogram and each sample histogram (lines 6–10). The lookup operation in line 7: *sHist[i].get(word)* is a bottleneck as it is called for each SFA word and sample. To have constant time lookups, we decided to implement each BOSS histogram as a hash map.

Parallel Execution (Algorithm 5.2): We do not parallelize a single *predict*-method-call. Instead, when running multiple queries, each 1-NN query (*predict*-method-call) runs in a separate thread. This workload is embarrassingly parallel.

Fit (Algorithm 5.3): The *fit*-method has to find the optimal set of parameters for the window length w, and the SFA word length l. We empirically observed that a constant SFA alphabet size of $c = 4$ was optimal throughout most datasets. This observation is in line with other works [47, 48]. Thus, we c fixed to four symbols. The *mean_norm*-parameter is a Boolean parameter, which is constant for a whole dataset as opposed to

Algorithm 5.3 Fit: Train the model using leave-one-out cross-validation.

```
1  [(int,int,int,map<String,int>)] fit(TimeSeries[] samples, bool mean_norm)
2    [(int,int,int,map<String,int>)] scores = []
3    int maxL=16
4    int c=4
5    int minW = 10
6    // search all window lengths
7    for int w = maxW down to minW   // in parallel
8      // obtain histograms for maxL
9      map<String, int>[] sHist = []
10     for int i in 1..len(samples)  // in parallel
11       sHist[i] = BOSSTransform(samples[i], w, maxL, c, mean_norm)
12     // search all SFA word lengths
13     int bestCor=0, int bestF=0
14     for int l in [8,10..maxL]
15       map<String, int>[] bags = createHistogram(sHist, f)
16       int correct=0
17       for int qId in [1..len(samples)]  // leave-one-out, in parallel
18         String nnLabel = predict(bags[qId], bags)
19         if (nnLabel==samples[qId].label) correct++
20       // store best feature per window length
21       if correct > bestCor
22         bestCor = correct
23         bestF = f
24     // store best scores for each window length
25     scores.push((bestCor, w, bestF, sHist))
26   return scores
```

setting it per sample or window. If set to *true*, the first Fourier coefficient (DC coefficient) is dropped to obtain offset invariance.

The algorithm performs a grid-search using leave-one-out cross-validation on the train samples. The parameter space is given by $mean\_norm \epsilon [true, false]$, *window length* $w \epsilon [10, n]$, and *SFA word length* $l \epsilon \{8, 10, 12, 14, 16\}$. In total this parameter space has the size: $n \cdot 5 \cdot 2 = O(n)$.

All window lengths (lines 7ff) are iterated to obtain the optimal SFA word length (lines 14ff). The BOSS histograms are constructed using the longest word length $maxL = 16$ (lines 10–11). Shorter word lengths are then tested by dropping the rear symbols of each SFA word and rebuilding the histogram (line 15) (described in more detail in Chapter 5.5.1). In case of an accuracy tie between two SFA word lengths, the smaller word length is kept (lines 21–23). This follows the assumption that a stronger noise-reduction is generally preferable. Finally, the accuracy scores for each pair of window length and SFA word length are collected (line 25) and returned (line 26). We will use these for our ensemble classifier in the next section.

Parallel Execution (Algorithm 5.3): The training of the classifier is well-suited for parallel execution: There is no data-dependency among window lengths in line 7. There are two nested embarrassingly parallel regions in lines 10–11 for each histogram and in lines 17–19 for each 1-NN search.

5.4.2 Computational Complexity

Prediction (Algorithm 5.2): The computational complexity of the *predict*-method is given by the transformation of the query and a 1-NN search over the N train samples using the BOSS distance calculations for window length w:

$$\begin{aligned} T(BOSS\ Predict) &\epsilon O(T(BOSS) + N \cdot T(BOSSDistance)) \\ &\epsilon O(n + w \log w + Nn) \\ &= O(Nn + w \log w) \end{aligned}$$

It has a significantly lower computational complexity than the Shotgun distance with $O(Nn^2)$. We postpone a detailed comparison of the computational complexity of the rivaling methods to Chapter 6.

Fit (Algorithm 5.3): The computational complexity of the *fit*-method results from leave-one-out cross-validation in combination with the 1-NN search. To obtain the best window lengths, at most $O(n)$ window lengths have to be tested to predict N labels each. Leave-one-out cross-validation has a quadratic computational complexity in the number of samples N:

$$\begin{aligned} T(BOSS\ Fit) &\epsilon O(N \sum_{w=1}^{n} [T(BOSS) + T(BOSS\ Predict)]) \\ &\epsilon O(Nn^2 + N^2n^2 + N \sum_{w=1}^{n} w \log w) \\ &= O(N^2n^2 + N \sum_{w=1}^{n} n \log n) \\ &= O(N^2n^2 + Nn^2 \log n) \end{aligned}$$

We omit the SFA word length, as it contains only five values to be tested and is thus constant in the length of the time series. We postpone a detailed comparison of the computational complexity of the rivaling methods to Chapter 6.

5.4.3 BOSS Ensemble Classifier Algorithm

The BOSS ensemble classifier (Algorithm 5.4) follows the same intuition as the Shotgun ensemble classifier. It extends the BOSS classifier (Algorithm 5.2) by representing each time series by multiple window lengths to allow for varying lengths of characteristic patterns. We skip the details and refer to Chapter 4.4.3 for details. In our experiments, factors in between $[0.92, 9.05]$ were best throughout most datasets.

Algorithm 5.4 Predict: The BOSS Ensemble Classifier.

```
1  String predictEnsemble(TimeSeries query, [(int,int,int,map<String,int>)] scores, bool
       mean_norm, double factor)
2    int c=4
3    // stores for each window length a label
4    String[] windowLabels = []
5    int bestScore = max([ correct | ( correct ,_,_,_) in scores ])
6    // determine the label for each window length
7    for (correct, w, bestF, sHist) in scores // in parallel
8      if (correct > bestScore * factor)
9        // compute query histogram
10       map<String,int> qHist = BOSSTransform(query,w,bestF,c,mean_norm)
11       windowLabels[len] = predict(qHist,sHist)
12   return most frequent label from windowLabels
```

5.5 Search Space Pruning

The rationale of search space pruning is to early abandon unpromising candidates, if these cannot result in finding a new optimum. In our Shotgun distance model (Chapter 4.5) we aimed at early abandoning distance calculations and an upper bound on the accuracy for training a classifier. The BOSS makes use of these two optimizations, and as such these are not explicitly stated here again.

5.5.1 Incremental Refinement of SFA word lengths

Using smaller SFA word lengths has the effect of reducing noise but goes hand in hand with a loss in signal details. A core idea of our BOSS model is to test different SFA word lengths to find the optimal trade-off between word length and noise reduction by maximizing the classification accuracy on a train dataset. To avoid redundant computations we make use of the *frequency domain nature* of SFA (Chapter 3.3.1). Our rationale is:

1. Calculate the SFA transformation for the largest required SFA word length l (Algorithm 5.3 lines 9–11),
2. Incrementally add/remove the last character(s) from each SFA word, and
3. Update the BOSS histograms (Algorithm 5.3 line 15).

Given this optimization, we avoid recalculating all SFA words for each new word length, which would be equal to moving lines 9–11 to line 15.

5.5.2 Lower Bounding of larger SFA word lengths

An important observation is that the smaller SFA word length l_1 can be used to lower bound the distance computations on the larger SFA word lengths $l_2 > l_1$ to avoid unpromising computations. That means we can use the BOSS distance on SFA word length l_1 to decide, if we have to test a larger word length l_2.

Claim 2. Given alphabet size c, two BOSS histograms $B_{1;l_1}$and $B_{2;l_1}$ at word length l_1 and $B_{1;l_1+1}$ and $B_{2;l_1+1}$ at word length l_1+1, the following applies:

$$\frac{1}{c} \cdot dist(B_{1;l_1}, B_{2;l_1}) \quad \leq \quad dist(B_{1;l_1+1}, B_{2;l_1+1}) \tag{5.11}$$

Proof. Given any SFA word $a\epsilon\Sigma^{l_1}$, and the SFA words $(ab)\epsilon\Sigma^{l_1+1}$ derived from concatenating a of length l_1 with a symbol $b\epsilon\Sigma$, the following applies:

$$\frac{1}{c}\left[B_{1;l_1}(a) - B_{1;l_1}(a)\right]^2 \quad = \quad c\left[\frac{B_{1;l_1}(a)}{c} - \frac{B_{2;l_1}(a)}{c}\right]^2 \qquad (\textit{binomial theorem}) \tag{5.12}$$

Based on this, we claim that:

$$\underbrace{c\left[\frac{B_{1;l_1}(a)}{c} - \frac{B_{2;l_1}(a)}{c}\right]^2}_{(\overline{x}-\overline{y})^2} \quad \leq \quad \sum_{b\epsilon\Sigma} \underbrace{\left[B_{1;l_1+1}(ab) - B_{2;l_1+1}(ab)\right]^2}_{(x_i-y_i)^2} \tag{5.13}$$

$$\iff \qquad c(\overline{x}-\overline{y})^2 \quad \leq \quad \sum_{i=1}^{c}(x_i-y_i)^2 \tag{5.14}$$

That means, the square difference in the means is smaller or equal to the sum of square differences. This Equation 5.13 has been proven in [40]. Our proof of Equation 5.11 ends by extending Equation 5.12 to all SFA words in $B_{1;l_1}$:

$$\begin{aligned}
\frac{1}{c} \cdot dist(B_{1;l_1}, B_{2;l_1}) &= \frac{1}{c} \cdot \sum_{a\epsilon\Sigma^{l_1}} \left[B_{1;l_1}(a) - B_{2;l_1}(a)\right]^2 && (\textit{by Eq. 5.7}) && (5.15)\\
&= \sum_{a\epsilon\Sigma^{l_1}} c\left[\frac{B_{1;l_1}(a)}{c} - \frac{B_{2;l_1}(a)}{c}\right]^2 && (\textit{by Eq. 5.12}) && (5.16)\\
&\leq \sum_{a\epsilon\Sigma^{l_1}} \sum_{b\epsilon\Sigma} \left[B_{1;l_1+1}(ab) - B_{2;l_1+1}(ab)\right]^2 && (\textit{by Eq. 5.13}) && (5.17)\\
&= \sum_{a\epsilon\Sigma^{l_1+1}} \left[B_{1;l_1+1}(a) - B_{2;l_1+1}(a)\right]^2 && && (5.18)\\
&= dist(B_{1;l_1+1}, B_{2;l_1+1}) && && (5.19)
\end{aligned}$$

□

5.6 Experiments

5.6.1 Setup

We evaluate the BOSS model based on case studies for not pre-processed and noisy time series datasets and on the established UCR benchmark datasets [42] (see Appendix Table 8.1). The webpage accompanying the BOSS publication contains all raw numbers and

Dataset	State of the Art	1-NN DTW	1-NN BOSS ensemble classifier	
Anthropology (Arrowhead)	80% [91]	66.3%	88.6%	$factor : 0.95, mean : T$
Medicine (BIDMC)	92.4% [38]	62.8%	100%	$factor : 0.95, mean : F$
Security (Passgraph)	70.2% [52]	71.8%	74%	$factor : 0.95, mean : F$
Historical Document (Shield)	89.9% [91]	86%	90.7%	$factor : 0.95, mean : T$
Astronomy (StarlightCurves)	93.7% [63]	90.7%	97.6%	$factor : 0.95, mean : F$
Motions (Toe Segmentation)	91% [91]	66.2%	98.2%	$factor : 0.95, mean : T$
Spectrographs (Wheat)	72.6% [91]	71.3%	82.6%	$factor : 0.95, mean : T$

Table 5.1 – Test accuracies on the case studies.

the C++ source codes [72]. The BOSS ensemble classifier was implemented in JAVA and C++ using OpenMP. All experiments were performed using the JAVA implementation on a shared memory machine running LINUX with 8 Quad Core AMD Opteron 8358 SE processors, and JAVA JDK x64 1.8. For all experiments the time series datasets were z-normalized prior to the experiments (compare Chapter 2.2). All experiments consist of two phases: model building using the train dataset and testing the classification accuracy using the test dataset.

Each dataset provides a *train/test* split. There might be a confusion with the terms "training set", "validation set", and "test set" used in supervised learning. The train split is used as the training and validation set. The test split is used as the test set. By the use of these train/test splits, the results are comparable to previous publications. All our results are based on the *test accuracy* of the classifiers using the test split.

5.6.2 Case Studies: Hierarchical Clustering and Classification

Several kinds of invariances for distance measures were presented (compare Chapter 2.2). The BOSS model accounts for the following invariances:

1. Amplitude/offset, due to *normalization* of the windows,
2. Local scaling, due to *windowing* the query and sample,
3. Phase shifts, due to the *BOSS model* (unordered set of SFA words),
4. Occlusion, due to the *BOSS distance* (tolerance to missing/extraneous data), and
5. Noise, due to the noise reducing properties of *SFA*.

This is confirmed by the following case studies. A dendrogram plot (binary tree) is used to illustrate the results of a hierarchical clustering. The dendrogram makes use of u-shapes

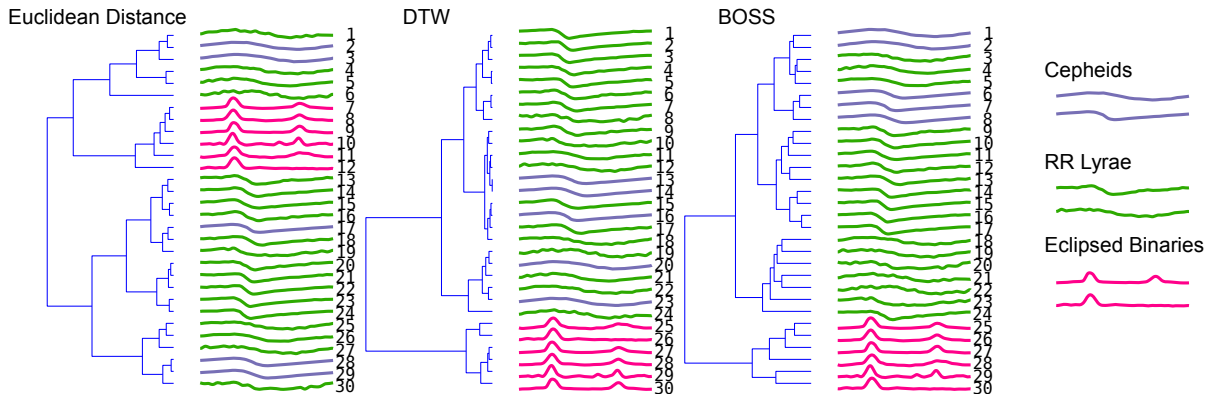


Figure 5.7 – Dendrogram of a hierarchical clustering of the StarlightCurves dataset. There are three types of star objects: *Eclipsed Binaries*, *Cepheids*, and *RR Lyrae Variables*.

that connect the two most similar time series. The height of each u-shape is equal to the distance (similarity). We set the parameters of the BOSS model using cross-validation on the train datasets.

Astronomy

It is possible to get large amounts of data, but it can be very time consuming to obtain labels for each data item. Thus, it is difficult to obtain large amounts of labeled data. The *StarlightCurves* dataset is one of the largest freely available datasets [42] that consists of $N = 1000$ train and $N = 8236$ test starlight curves, each of length $n = 1024$. There are three types of star objects: *Eclipsed Binaries* (purple), *Cepheids* (blue), and *RR Lyrae Variables* (green). This dataset is of particular interest as there are dozens of papers referencing it.

Hierarchical Clustering: Figure 5.7 illustrates a dendrogram of a hierarchical clustering of the data. The *Cepheids* and *RR Lyrae Variables* have a similar shape and are thus difficult to separate. Both, the ED and DTW result in visually unpleasing clusterings, as they fail to separate these two classes. The BOSS performs best in separating the two classes, which is a result of local scaling (windowing), the noise reduction of SFA, and the phase invariance of the BOSS model.

Classification: The 1-NN BOSS ensemble classifier outperforms previous approaches in terms of classification accuracy with a test accuracy of 97.6% (Table 5.1): The 1-NN DTW classifier achieves a test accuracy of 90.7%, and the highest reported test accuracy in literature is 93.7% [63].

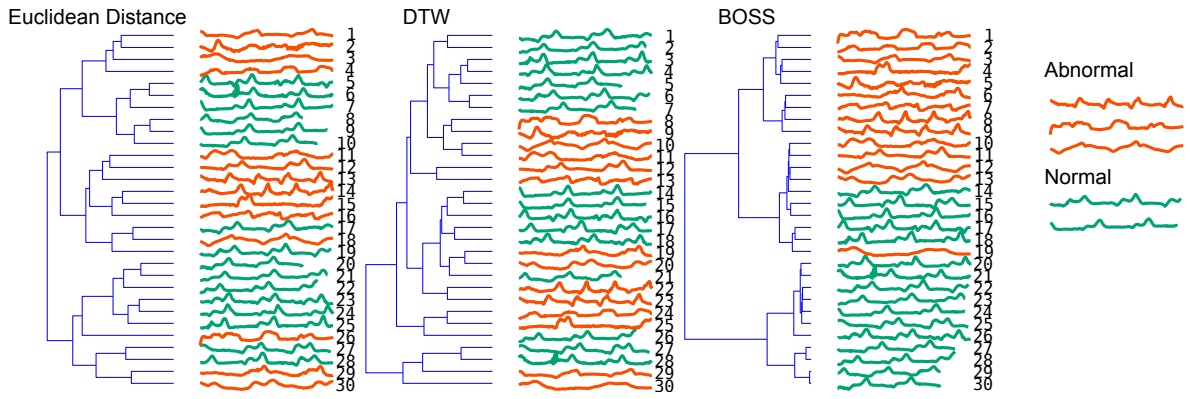


Figure 5.8 – Dendrogram of a hierarchical clustering of human walking motions. There are two motions: *Normal* (green) and *Abnormal* (orange) walk.

Human Walking Motions:

The CMU [26] contains walking motions of four subjects. The authors [91] provide multiple segmentation approaches and we used their first segmentation approach. Each motion was categorized by the labels *normal walk* (green) and *abnormal walk* (orange). The data were captured by recording the z-axis accelerometer values of either the right or the left toe. The difficulties in this dataset result from variable length gait cycles, gait styles and paces due to different subjects throughout different activities including stops and turns. A normal walking motion consists of up to three repeated gait cycles.

Hierarchical Clustering: Figure 5.8 shows a dendrogram of a hierarchical clustering of the walking motions. The ED fails to identify the abnormal walking styles, thus these are not clearly separated from the normal walking motions. DTW provides invariance to phase shifts by a peak-to-peak and valley-to-valley alignment of the time series. This still does not result in a satisfying clustering as the abnormal and normal walking patterns are intermingled. As part of our BOSS model the patterns from the walking motions are extracted and noise reduction is applied. As a result, the separation of the normal walking motions from the abnormal walking motions is close to perfect with just the 19th walking motion being out of place.

Classification: The 1-NN DTW classifier gives a test accuracy of 66%. The best reported accuracy in literature [91] is 91%. The 1-NN BOSS ensemble classifier provides a test accuracy of 98.2% (Table 5.1). This is by far the best reported accuracy.

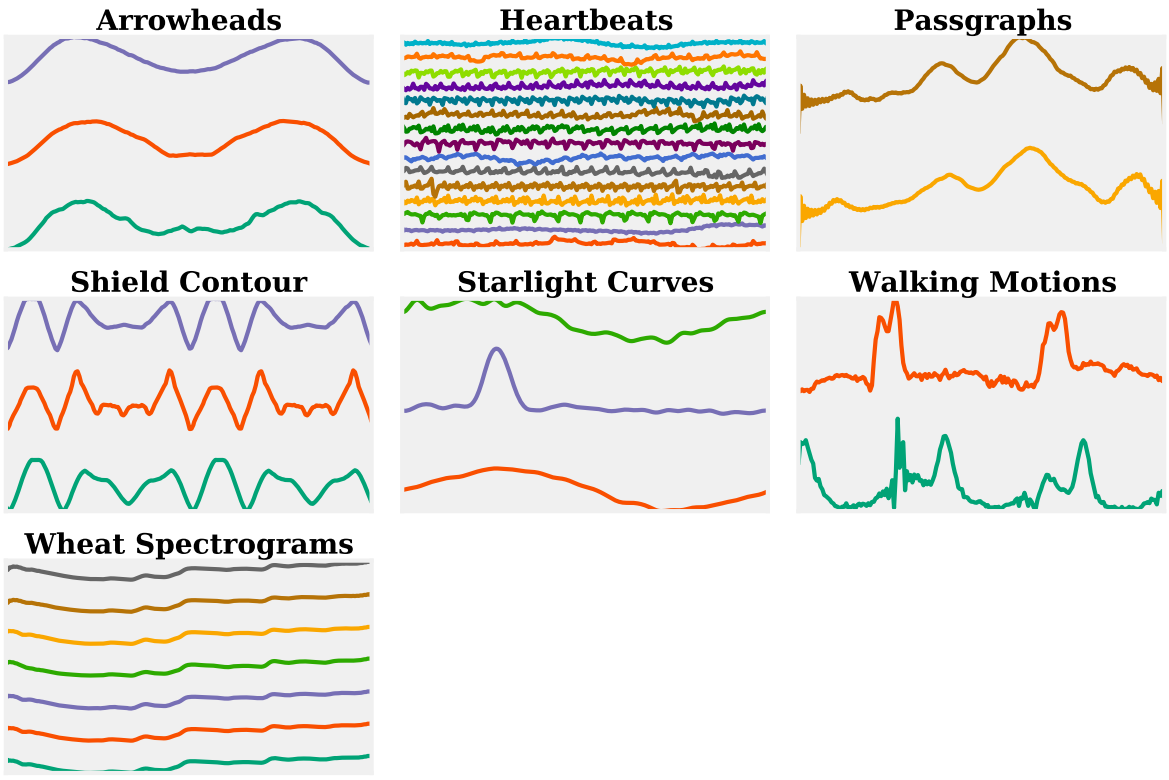


Figure 5.9 – One sample for each class of the seven case studies.

Anthropology, Historical Documents, Personalized Medicine, Spectrography and Security

We complement the case studies using datasets covering personalized medicine, anthropology, historical documents, mass spectrography and security (Figure 5.9).

- *Passgraph* [52] represents grids of dots, which a user has to connect to gain access to a resource like his smartphone.
- *Arrowheads* [90] is a dataset representing the shape of projectile points of variable lengths.
- *Shield* [90] contains heraldic shields of variable lengths.
- *Wheat* [90] is a dataset of spectrographs of wheat samples grown in Canada clustered by year.
- The *BIDMC Congestive Heart Failure Database* [3] is a dataset that contains ECG recordings (heartbeats) of different subjects. These suffer from severe congestive heart failures.

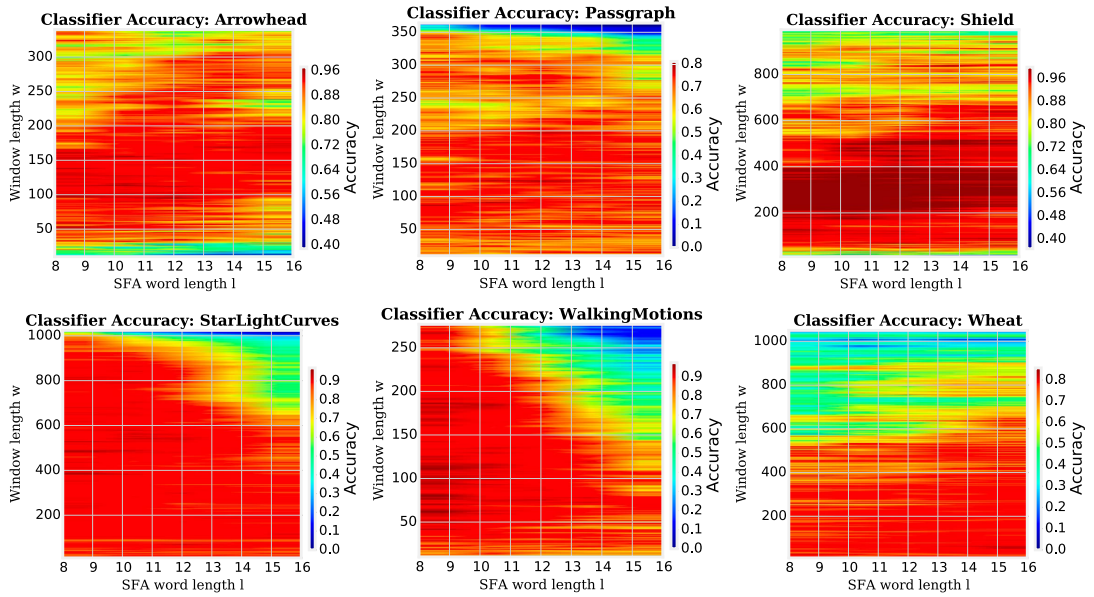


Figure 5.10 – The parameter space of the BOSS model on the case studies for different SFA word lengths and window lengths.

The results in Table 5.1 show that the BOSS ensemble classifier improves the accuracy on a large scope of application areas including raw, extraneous, erroneous, and variable length data. It performs significantly better than the best, specialized rivaling methods by up to 10 percentage points. The accuracy gap to 1-NN DTW is up to 37 percentage points.

Implications: The BOSS model provides invariances to amplitude, local scaling, phase shifts, occlusion, and noise. As such it outperforms the state of the art on several case studies in terms of classification and clustering accuracy.

Parameter Space

Figure 5.10 shows the parameter space for the presented case studies. It shows that the BOSS model is surprisingly robust to the choice of the two parameters *window length* and *SFA word length*. A small SFA word length is generally preferable, which correlates to a strong noise reduction (low-pass filter). The window length depends on the length of the characteristic structures in the dataset. As such, the length of a gait cycle in the walking motion dataset is roughly equal to a window length of 120 (compare Figure 4.2). In case of too large window lengths or too little noise reduction, the BOSS model quickly degenerates.

Implications: For the BOSS model a small SFA word length is advantageous, which is equal to a strong noise reduction. The window length parameter is dataset dependent. The BOSS ensemble classifier helps to mitigate the effects of choosing a bad window length, by using multiple window lengths.

Scalability

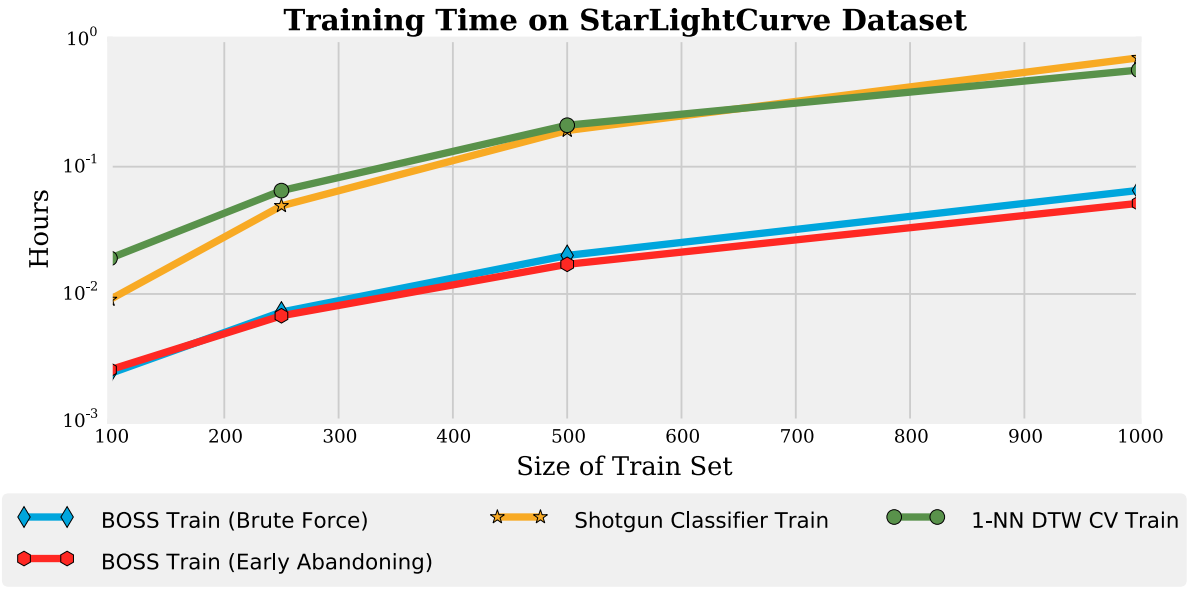


Figure 5.11 – The time required to for training on *StarLightCurves* dataset using the presented pruning strategies.

To test the scalability of the BOSS *fit*-method, we iteratively doubled the number of samples from $N = 100$ to $N = 1000$, each of length 1024, and measured the pruning strategies presented in this thesis:

- Brute Force (Algorithm 4.3),
- Early abandoning using the Lower Bounding of larger SFA word lengths (Chapter 5.5.2),
- The Shotgun Distance Classifier, and
- 1-NN DTW CV with a warping window constraint set through cross-validation using the state of the art implementation [62] (compare Chapter 2.2).

All reported numbers were measured using 32 cores. Figure 5.11 shows that the train time of the *brute force* algorithm grows quadratically in N to approximately four minutes for $N = 1000$ samples. *Early abandoning* reduces this by 25% to three minutes. The

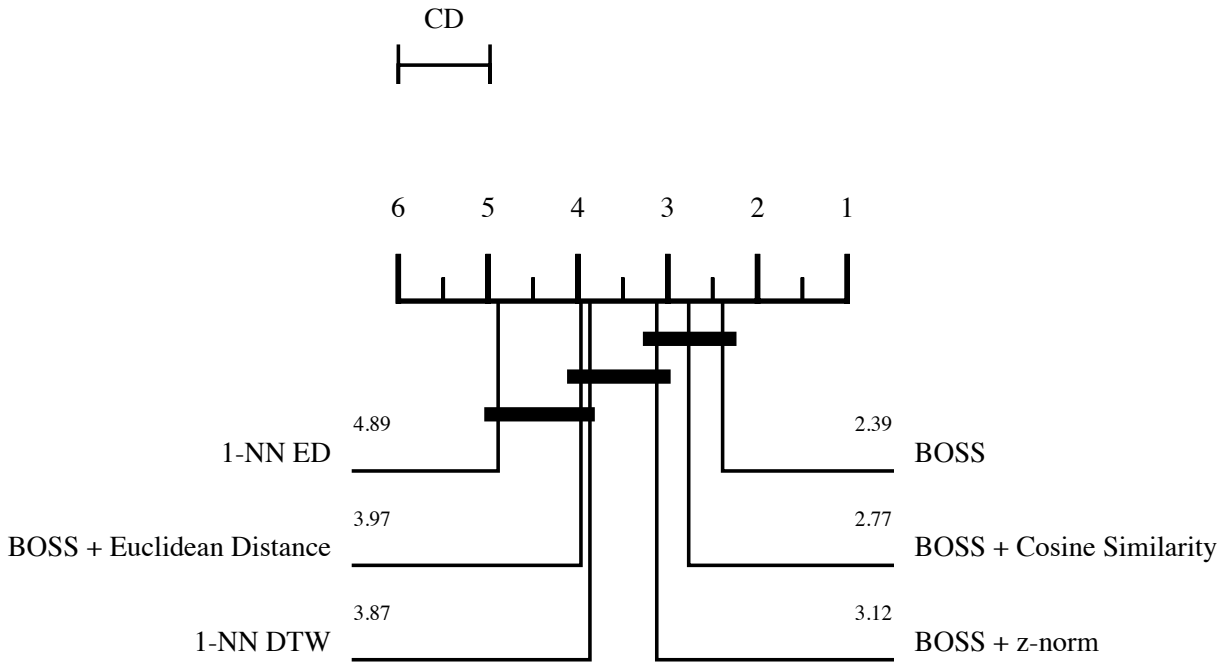


Figure 5.12 – Critical difference diagram for the design decisions made using the 45 UCR datasets. The best classifiers are to the right. Critical difference is 1.02.

resulting train time is one order of magnitude lower than that of the 1-NN DTW CV classifier or the 1-NN Shotgun classifier.

Implications: The presented pruning strategies reduce the train time by up to 25%. The BOSS ensemble classifier has a one order of magnitude lower train time than the 1-NN DTW CV classifier on this dataset. We postpone the more detailed runtime analysis to Chapter 6.

5.6.3 Impact of Design Decisions

We use all 46 UCR datasets [42] to test the impact of our design decisions (Table 8.1). The BOSS ensemble classifier is based on two design decisions:

1. The BOSS model using the *BOSS distance* as opposed to
   (a) the Euclidean distance (*BOSS + Euclidean distance*) or
   (b) the Cosine similarity (*BOSS + Cosine similarity*).
2. *Mean normalization* as a parameter as opposed to applying z-normalization *(BOSS + z-norm*) always to all windows.

We do not repeat the test of the effects of ensemble classification, as this has been shown in Chapter 4.3.2.

Figure 5.12 shows a critical difference diagram over the average ranks of the classifiers as introduced in [28]. The classifiers with the lowest (best) ranks are to the right. The group of classifiers that are not significantly different in their rankings are connected by a bar. The critical difference (CD) length is shown above the graph.

Overall the presented BOSS model using the BOSS distance and mean normalization as a parameter shows the best (lowest) rank. Using the Cosine similarity as a distance measure or always norming the mean value performs significantly worse.

Implications: The BOSS model is based on the BOSS distance, and mean normalization as a parameter. Both improve the classification accuracy, as the mean can be characteristic for the similarity of patterns and providing tolerance to extraneous data is beneficial.

5.6.4 Classification Accuracy Benchmark

The comparison to state of the art classifiers with regards to the classification accuracy and classification times will be presented in Chapter 6.

5.7 Summary

The time series data analytics task is complicated by extraneous, erroneous, and unaligned data of variable length. However, the extraneous or erroneous data, as a result of noise, have been paid surprisingly little attention to. Noise is commonly assumed to be filtered as part of a pre-processing step carried out by a human. Our BOSS model is a structure-based similarity model for time series based on the bag-of-patterns model and the SFA representation. Subsequences are extracted from a time series, low-pass filtering and quantization is applied to each substructure using the SFA transformation, and a histogram of SFA words is built. Two time series are then compared based on the differences in the histogram of SFA words, which is equal to the substructural differences. The approach makes the BOSS model alignment-free and robust to noise. Optimization techniques are presented to reduce the computational complexity of the BOSS ensemble classifier up to the level of Dynamic Time Warping while being significantly more accurate. The BOSS ensemble classifier is based on 1-NN classification and represents each time series by multiple BOSS models at different substructural sizes. As part of our experimental evaluation we show that in the context of hierarchical clustering and classification the BOSS model performs significantly better than state of the art.

The BOSS model is the most accurate similarity model for time series data analytics to this day. It combines algorithms from the Shotgun Distance model with the symbolic representation SFA. The main drawback is its computational complexity, which makes it

unfeasible for analyzing large time series datasets. In the next chapter we address the scalability of the BOSS model.

Chapter 6

Bag-Of-SFA-Symbols in Vector Space: Scalable Time Series Classification

The BOSS VS model is the final time series model presented in this thesis (compare Figure 1.2). This chapter gives a detailed description of the BOSS VS model. It aims at scalability in combination with alignment-free and noise-robust classification. Parts of this chapter have been published [74].

6.1 Introduction

In the last decades there has been an enormous increase in data volumes which are expected to reach 100 zettabytes (10^{21}) by 2020 [85]. At the same time, sensors for recording time series have become cheap and omnipresent as in RFID chips, wearable sensors (wrist bands, smartphones), smart homes (smart plugs [39], smart meters, fire alarms, temperature, security), or event-based systems (soccer player shoes [53]). A smart-meter with a sampling rate of one value per minute records more than 0.5 million values a year. Given a company with millions of customers, this easily accounts for trillions (10^{12}) of measurements a year. The goal is to extract knowledge from that raw data.

The availability of the UCR time series benchmark datasets [42] has led to a wealth of time series classification algorithms. However, the over-dependence on these datasets is troublesome as the largest of those datasets (StarlightCurves) has a several thousand time series. We believe that the computational complexity of most state of the art classifiers (compare Table 6.1) is excessive in terms of train or test times even when it comes to moderately sized datasets of $N \geq 10^3$ time series. For many classifiers the computation complexity has been reduced by the use of early abandoning and cascading lower bounds (compare Table 6.1). Bagnall et al. [13] claim:

> "A new algorithm is only of interest in terms of accuracy if it can significantly outperform 1-NN DTW with a full warping window".

	Train Complexity	Test Complexity		Accuracy
		lower bound	upper bound	
1-NN ED [62]	-	$\Omega(n+N)$	$O(Nn)$	
SAX-VSM [81]	$O(Nn^3)$	$O(n)$		
Shapelets [52, 63]	$O(N^2n^3)$ to $O(Nn^2)$	$O(n)$		
1-NN DTW [13, 62]	-	$\Omega(n^2+N)$	$O(Nn^2)$	+
1-NN DTW CV [13, 62]	$O(N^2n^2)$	$\Omega(nr+N)$	$O(Nnr)$	+
SVM	$O(N^2n)$ to $O(N^3n)$	$O(n)$		+
Shotgun Ensemble [71]	$O(N^2n^3)$	$\Omega(n^2+N)$	$O(Nn^2)$	+
PROP [13]	ensemble; depends on used classifiers			++
1-NN BOSS [68]	$O(N^2n^2+Nn^2\log n)$	$\Omega(n+N)$	$O(Nn+w\log w)$	++
1-NN BOSS VS	$O(Nn^{\frac{3}{2}}\log n)$	$O(n+w\log w)$		+

Table 6.1 – Computational complexity (upper $O(...)$ and lower bounds $\Omega(...)$) of state of the art classifiers for n=length, w=window length (typically $w \ll n \ll N$), N=number of time series, r=warping window size constraint. Accuracy score is based on the ranking on the 91 datasets in Figure 6.7.

In another publication, Bagnall et al. state that "accuracy is, in our option, the most important [metric]" [49]. We believe that the key challenge for time series data analytics is to provide scalability in train and test times while maintaining high accuracy and claim:

> A new algorithm is of interest if it can significantly outperform 1-NN DTW with a full warping window in terms of *accuracy* OR *classification times*.

Figure 6.1 shows the cumulative CPU time each classifier took for training and testing on all 91 datasets (see Appendix Table 8.1). It is the total time needed to run all tests on a single core. In these datasets there are roughly $N = 50000$ train and $N = 100000$ test time series. We benchmarked the most accurate classifiers according to the experiment in Figure 6.7. PROP is an ensemble of classifiers including 1-NN DTW with a warping window constraint set through cross validation (1-NN DTW CV) and as such cannot be faster.

The slowest classifier 1-NN DTW CV, using the state of the art implementation [62], takes more than **2000 CPU hours** (!) until completion for all datasets. 1-NN DTW does not require training and finishes within **80 CPU hours**. The most accurate 1-NN BOSS ensemble classifier (Chapter 5) finishes after roughly **97 CPU hours**. Our 1-NN Bag-Of-SFA-Symbols in Vector Space (BOSS VS) classifier outperforms the other classifiers by one to three orders of magnitude in terms of time to completion. It takes roughly 7 **CPU**

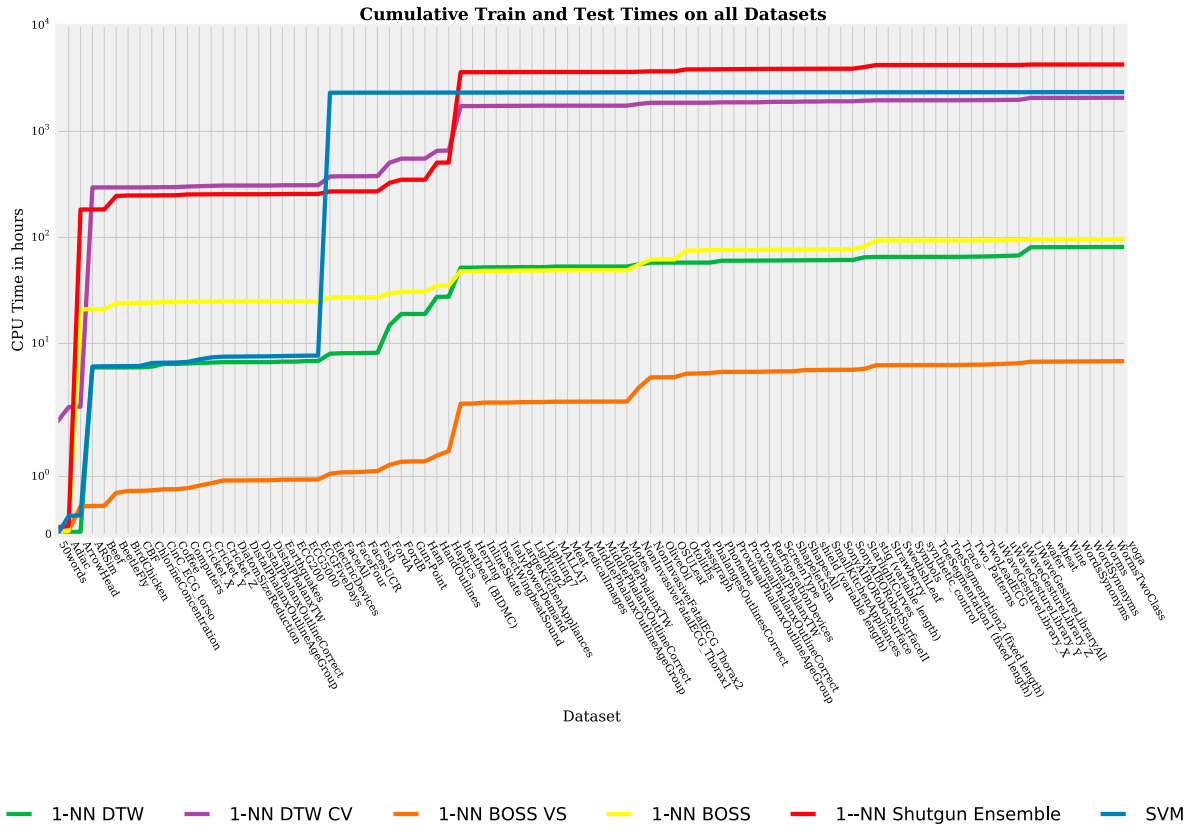


Figure 6.1 – Cumulative train and test times on all 91 datasets (Appendix Table 8.1) using a single core. Datasets with enormous execution times are the StarLightCurves, Heartbeat, or ElectricDevices.

hours for all datasets. These empirical results are not surprising given the complexities of the other classifiers in Table 6.1.

To put a factor of 10^3 into relation: we can solve a classification problem with 1-NN DTW CV that runs on a cluster of 4000 cores for one day [13], with 1-NN BOSS VS using commodity hardware and a 4 core cpu within one to two days resulting in a similar or better classification accuracy.

The BOSS model (Chapter 5) highlights the importance of tolerance to noise for a similarity measure. In this chapter, we significantly reduce the computational complexity of the BOSS model to support the classification of larger time series datasets within seconds to minutes where other accurate classifiers require hours to days (compare Table 6.1 and Figure 6.1). The BOSS VS model extends the BOSS model by a compact representation of classes instead of time series, which significantly reduces the computational complexity, weights characteristic SFA words, but trades off reduced accuracy for improved train times. The 1-NN BOSS VS has a test complexity of $O(n)$ which allows for the classification of massive time series datasets. Its moderate train complexity of $O(Nn^{\frac{3}{2}})$ (that is lower than the test complexity of 1-NN DTW) allows for frequent model updates as in real-time

predictive analytics.

The 1-NN BOSS VS is not the most accurate classifier, but it is (a) multiple orders of magnitude faster than the 1-NN BOSS ensemble classifier, 1-NN DTW, the Shotgun ensemble classifier, and state of the art, and (b) it is significantly more accurate than 1-NN DTW with or without a warping window constraint. Our contributions are as follows:

- This is the first work on time series classification we are aware of that focusses on the computational complexity and compares state of the art classifiers regarding classification times on 91 public time series datasets.
- We present the 1-NN BOSS VS classifier that combines the noise tolerance of the BOSS model with fast train and test times due to the use of the vector space model in Chapter 6.3.
- In our experimental evaluation, we present two case studies for moderately sized time series datasets which illustrate that the BOSS VS classifier is orders of magnitude faster than state of the art classifiers in Chapter 6.5.2.
- We present an exhaustive study using 91 time series datasets, which shows that the 1-NN BOSS VS classifier is significantly more accurate than 1-NN DTW with or without a warping window constraint, while providing up to 4 orders of magnitude lower classification times than state of the art (Chapter 6.5.3).
- We show in Chapter 6.5.3 that the 1-NN BOSS VS classifier is the fourth best classifier with regards to classification accuracy. Our 1-NN BOSS ensemble classifier (Chapter 5) shows the best classification accuracy, followed by an ensemble technique PROP, and the 1-NN Shotgun Ensemble classifier (Chapter 4).
- Finally, we show the impact of our design decisions to the BOSS VS model (Chapter 6.5.5).

6.2 Background and Related Work

Classical data mining algorithms like SVMs, decision trees, rotation or random forests have been used in the context of time series [30]. However, these did not perform better than the 1-NN DTW classifier, which is commonly used as the benchmark to compare to [13, 49, 29]. Its computational complexity is quadratic in the time series length n and linear in the size N of the training dataset: $O(Nn^2)$. Much effort has been spent to reduce the computational complexity by the use of early abandoning techniques and cascading lower bounds to prune off unpromising candidates. These lower bounds have constant to linear time to compute with an increasing tightness of lower bounds. The UCR suite [62] is the state of the art implementation of 1-NN DTW. Other approaches include (compare Table 6.1):

- Most recently a 1-NN DTW nearest centroid classifier has been presented [56] to reduce the test complexity of 1-NN DTW to $O(n)$ at the cost of an excessive training complexity.

- The 1-NN DTW CV classifier sets a warping window constraint through cross-validation. This reduces the test complexity to $O(Nnr)$ for warping window constraint $r \leq n$, at the cost of an excessive train complexity of $O(N^2n^2)$.

- The PROP classifier [49] is one of the most accurate time series classifiers. It builds an ensemble of 11 classifiers including 1-NN DTW CV, 1-NN DTW, 1-NN LCSS, 1-NN ED, etc. As such, it has to train and run all of these classifiers to predict a label. It has the same computational complexity as the slowest used classifier.

- Shapelet classifiers [52, 63] extract representative variable-length subsequences from time series and use a decision tree for classification. These classifiers have a high computational complexity for training of $O(N^2n^3)$ [52] to $O(Nn^2)$ [63] and give a poor accuracy.

- The 1-NN Shotgun ensemble classifier [71] divides the query into disjoint subsequences and slides each query window over the sample to find the position that minimizes the Euclidean distance. The Shotgun classifier has a high $O(N^2n^3)$ complexity for training. Both, Shapelet and the Shotgun Classifiers are based on the Euclidean distance and are therefore sensitive to noisy data.

- The *bag-of-patterns* (BOP) model [48] is the closest to our BOSS model. BOP extracts substructures as higher-level features of a time series. BOP transforms these substructures using SAX for quantization and the Euclidean distance as similarity metric.

- SAX-VSM [81] is the successor of the BOP model. It extends the BOP model by the use of the *tf-idf* weighing of the bags and Cosine similarity as similarity metric. It uses one bag of words for each class, instead of one bag for each sample. SAX-VSM has a huge parameter space of $O(n^2)$ parameters and has to recalculate all SAX coefficients for each new set of parameters, as mentioned in the comparison of SFA and SAX in Chapter 1.1.1. This results in an unreasonably high train time of $O(Nn^3)$.

In contrast our 1-NN BOSS VS has a small parameter space with just $O(\sqrt{n})$ parameters, leading to very low train times, and uses a compact representation of classes, leading to a low test complexity of just $O(n)$.

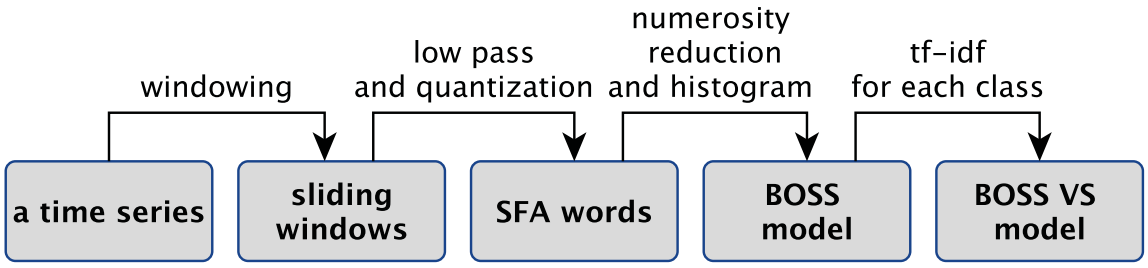


Figure 6.2 – The BOSS VS workflow: From a time series to the BOSS VS model.

6.3 The BOSS in Vector Space: An Alignment-free, Noise-Robust, and Scalable Similarity Model

6.3.1 Motivation

The BOSS VS model combines the BOSS model with the vector space model. The vector space model has first been introduced in information retrieval for representing text documents as vectors of keywords. Vector operations like Cosine similarity are used to compare the similarity of documents. Since then it has been applied to many other domains like audio [11] or time series retrieval [48, 81]. In the vector space model as proposed in [66] the *term frequency inverse document frequency* (*tf-idf*) model is used. The term frequency refers to the occurrence of words in a document. The *tf-idf* measure is used to weigh the term frequencies in the vector to give a higher weight to representative terms (words) of a class. In our model the *term* is an SFA word and a *document* corresponds to a time series.

Figure 6.2 illustrates the BOSS VS model. First, a time series is transformed to its BOSS model. Next, a tf-idf vector is computed for each class label, as opposed to each time series. The tf-idf vector serves as a model for each class.

Compared to other approaches the BOSS VS model has several advantages:

1. It is *alignment-free* as it extracts substructures and compares two time series based on their structural similarity;

2. It applies *noise reduction* by the use of SFA;

3. It provides invariances to phase shifts, local scaling, offsets, amplitudes, and occlusions;

4. It highlights characteristic SFA words by the use of the *tf-idf* weight matrix. SFA words that occur frequently across all classes are given a lower *idf* weight. This adds to the occlusion invariance and has a noise reducing effect;

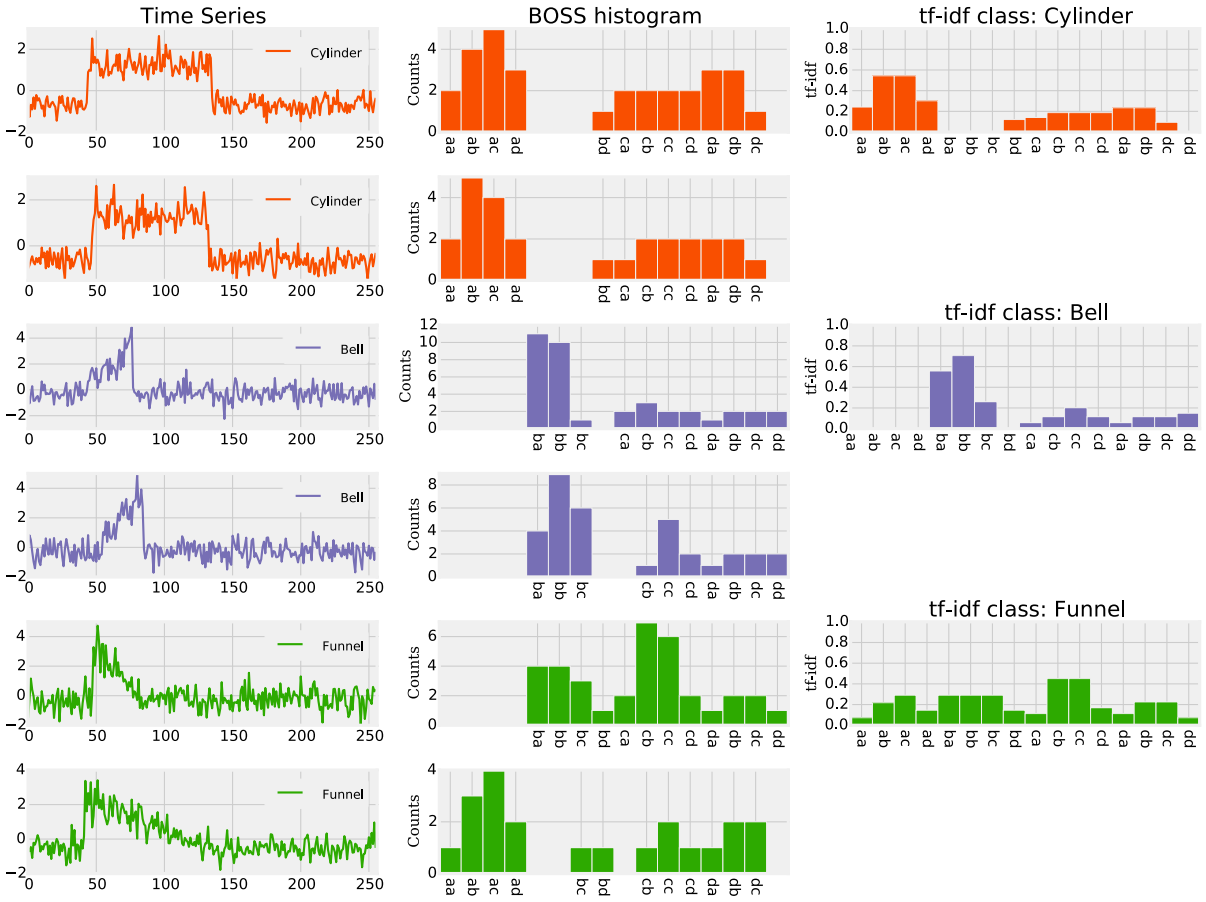


Figure 6.3 – Columns from left to right: The SFA word frequencies are stored in the BOSS histograms for each time series (center). Next, the *tf-idf* vectors are constructed for each class (right).

5. It uses a compact representation of classes instead of time series and thereby minimizes the influence of erroneous and extraneous data within a single time series, and significantly reduces the computational complexity;

6. It is fast, as it has low test and train times;

Properties (1) to (3) were introduced with the BOSS model [68]. The BOSS VS model adds properties (4) to (6) to the BOSS model.

6.3.2 The BOSS VS Model

Our BOSS VS model (Figure 6.3) has the same four parameters as the BOSS model (compare Chapter 5.3.2):

- **The window length** $w \in \mathbb{N}$: Represents the size of the substructures.

- **Mean normalization** $mean \, \epsilon \, [true, false]$: Set to true for offset invariance.
- The **SFA word length** $l \, \epsilon \, \mathbb{N}$ and **alphabet size** $c \, \epsilon \, \mathbb{N}$: Used for low-pass filtering and the string representation.

First, each time series is transformed to its BOSS histogram. Next, the *tf-idf* matrix is constructed using all BOSS histograms. It contains one *tf-idf* vector for each class (Cylinder, Bell, Funnel in Figure 6.3).

We use an approach presented in [81] and calculate the *inverse document frequency* (*idf*) for *each class* as opposed to *each time series*. The term frequency (*tf*) for an SFA word t of a time series T is given by:

$$tf(t,T) \quad = \quad \begin{cases} 1 + \log(B_T(t)) & , if \; B_T(t) > 0 \\ 0 & , otherwise \end{cases} \tag{6.1}$$

with $B_T(t)$ being the BOSS histogram, which represents the frequency of an SFA word t in the specific time series T. In the same manner the *term frequency (tf)* for an SFA word t within a class C is given by:

$$tf(t,C) = \begin{cases} 1 + \log(\sum_{T \epsilon C} B_T(t)) & , if \; \sum_{T \epsilon C} B_T(t) > 0 \\ 0 & , otherwise \end{cases} \tag{6.2}$$

The *inverse document frequency (idf)* captures how relevant an SFA word is across all time series T within a class C:

$$idf(t,C) = \log \frac{|CLASSES|}{\underbrace{|\{C \, | T \epsilon C \wedge B_T(t) > 0\}|}_{number \; of \; classes \; that \; contain \; t}} \tag{6.3}$$

This *idf* for an SFA word represents the total number of classes divided by the number of classes this SFA word occurs in. A high *idf* value is obtained by SFA words that occur only in a specific class.

The *tf-idf* of an SFA word t within a class C is thus defined as:

$$\begin{aligned} tfidf(t,C) \quad &= \quad tf(t,C) \cdot idf(t,C) & (6.4) \\ &= \quad (1 + \log(\sum_{T \epsilon C} B_T(t))) \cdot \log \frac{|CLASSES|}{|\{C \, | T \epsilon C \wedge B_T(t) > 0\}|} & (6.5) \end{aligned}$$

High *tf-idf* weights are obtained by SFA words with a high frequency that occur only in a specific class. Thus, SFA words that are common within all classes receive a low weight and are thereby filtered out.

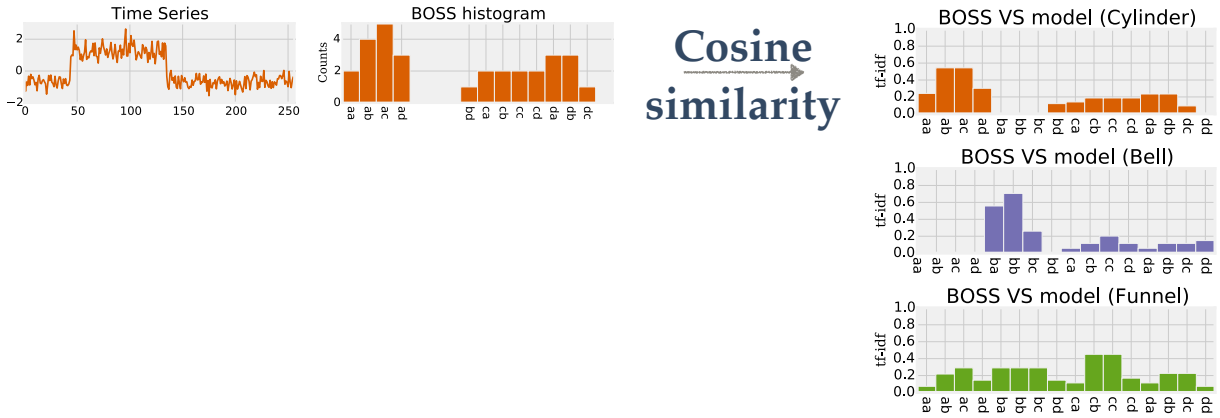


Figure 6.4 – 1-NN classification using the BOSS VS model: First, the query is transformed to its BOSS histogram, and finally the Cosine similarity is maximized in order to find the 1-NN to the query.

6.3.3 The BOSS VS Distance

The similarity of a *tf* vector of Q to an *tf-idf* class vector of C can then be computed using the Cosine similarity metric:

$$D_{BOSSVS}(Q,C) = \frac{\overrightarrow{Q} \cdot \overrightarrow{C}}{\left\|\overrightarrow{Q}\right\| \cdot \left\|\overrightarrow{C}\right\|} = \frac{\sum_{t \epsilon Q} tf(t,Q) \cdot tfidf(t,C)}{\sqrt{\sum_{t \epsilon Q}(tf(t,Q))^2}\sqrt{\sum_{t \epsilon C}(tfidf(t,C))^2}} \tag{6.6}$$

The lower bounding techniques introduced in the previous chapters are not applicable here, as we maximize the Cosine similarity, rather than minimizing it.

Computational Complexity: The computational complexity of the Cosine similarity is linear in the length of the time series n. Each BOSS histogram contains at most $n-w+1$ SFA words. A histogram lookup for an SFA word has a constant computational complexity of $O(1)$ by the use of hashing. This results in a total computational complexity that is linear in n:

$$T(BOSSVSDistance) \quad \epsilon \quad O((n-w+1)\cdot 1) \tag{6.7}$$

$$= \quad O(n) \tag{6.8}$$

While the computational complexity is upper bound by the time series length n, the actual number of *unique* SFA words is typically smaller due to duplicates and numerosity reduction.

6.4 Time Series Classification

Classification describes the task of assigning a label to an unlabeled time series Q using a trained model. Figure 6.4 illustrates this classification task. It requires the *tf-idf* weight

Algorithm 6.1 Predict: 1-NN classification using the BOSS VS model.

```
1  String predict(map<String,int> tf, map<String,int>[] tfIdfs)
2    (double maxSim, String bestClass) = (0, NULL)
3    for int classId in [1..len(tfIdfs)] // search classes
4      double cosSim = dotProduct(tf, tfIdfs[classId])
5      if cosSim > maxSim  // store the best class
6        (maxSim, bestClass) = (cosSim, classId)
7    return label(bestClass)
```

Algorithm 6.2 Fit: Train the parameters using leave-one-out cross-validation.

```
1  (int,int,int,map<String,int>) fit(TimeSeries[] samples, bool mean_norm, int[]
       windowLengths)
2    int maxL=16
3    int c=4
4    int bestCor=0, int bestL=0, int bestW=0
5    map<String,int>[] bestTfIdfs = []
6    // search all window lengths
7    for int w in windowLengths  // in parallel
8      // obtain the bags
9      map<String, int>[] sHist = []
10     for i in [1..len(samples)]  // in parallel
11       sHist[i] = BOSSTransform(samples[i], w, maxL, c, mean_norm)
12     // test all word lengths
13     for l in [4,6,8..maxL]
14       map<String, int>[] bags = createHistogram(sHist, f)
15       // tf-idf matrix
16       map<String,int>[] tfIdfs = calcTfIdf(bags, l)
17       int correct=0
18       for int qId in [1..len(samples)]  // leave-one-out, in parallel
19         String bestClass = predict(bags[qId], tfIdfs)
20         if bestClass has correct label then correct++
21       if correct > bestCor  // keep best
22         (bestCor, bestL, bestW, bestTfIdfs) = (correct, l, w, tfIdfs)
23   return (bestCor, bestL, bestW, bestTfIdfs) // return best parameters
```

matrix to be computed for each class (i.e., Cylinder, Bell and Funnel) based on a train dataset. First, the unlabeled query Q has to be transformed into the *tf* vector using the BOSS model and the optimal features (window length, mean normalization, SFA word length) obtained from the training phase. Next, the Cosine similarity is calculated using the *tf-idf* weight matrix for each class C of the train dataset. Finally the unlabeled time series is assigned to the class C that maximizes the Cosine similarity:

$$label(Q) = \underset{C \epsilon CLASSES}{argmax} \left(D_{BOSSVS}(Q, C)\right)$$

6.4.1 The BOSS VS Classifier Algorithm

Prediction (Algorithm 6.1): The algorithm is based on 1-Nearest-Neighbor (1-NN) classification using the *tf-idf* weight matrix. The use of the classes instead of the time series significantly reduces the computational complexity for classification. First, all classes (line 3) are iterated and the cross product between the *tf* vector of the query and

the *tf-idf* weight matrix for each class is calculated (line 4). The class that maximizes the Cosine similarity is chosen as the query's class label (lines 5–6).

Parallel Execution (Algorithm 6.1): We do not parallelize a single *predict*-method-call. Instead, when running multiple queries, each 1-NN query (*predict*-method-call) runs in a separate thread. This workload is embarrassingly parallel.

Fit (Algorithm 6.2): Training the BOSS model aims at finding the parameters that maximize the accuracy on a train dataset. We use grid search in combination with cross-validation to optimize the parameters for *mean*, w, l, and use a fixed alphabet size of $c = 4$. We have observed empirically that a constant alphabet size of 4 was sufficient for a high classification accuracy. The mean normalization is a boolean parameter, and increases the complexity by a constant factor of two when both *true* and *false* are tested. If set to *true*, the first Fourier coefficient (DC coefficient) is dropped to obtain offset invariance.

We choose the SFA word lengths from $\{4, 6, 8, 10, 12, 14, 16\}$, in total seven values. Searching for the optimal window length can be computationally expensive, as there are at most n windows for time series length n. To significantly reduce training times, we decided to train using only the $\sqrt{n}$ windows at equivalent distance in the interval $[10, n]$. As the BOSS model is robust regarding the choice of window lengths (Chapter 5.6.2), this has no significant effect on the accuracy. In total this parameter space has the size: $2 \cdot 7 \cdot \sqrt{n} = O(\sqrt{n})$. We will analyze the effects of these design decisions in Section 6.5.5. For comparison, SAX-VSM has a parameter space of $\mathrm{O}(n^2)$, as it tests $O(n)$ window lengths, $O(n)$ word lengths, 2 to 16 symbols, and three numerosity reduction schemes. The BOSS model (Chapter 5) has a parameter space of size $O(n)$, leading to higher train times.

At the end of the training phase, we obtain *tf-idf* vectors for each class of the train dataset. This *tf-idf* matrix is the compressed representation of the train dataset and used as the model for classification.

Algorithm 6.2 iterates all $\sqrt{n}$ window lengths (line 7) and obtains the BOSS model for each window length (lines 10–11). Next, the *tf-idf* weight matrix is computed for each class based on the BOSS models of each time series and a concrete SFA word length (lines 14–16). It uses the frequency domain property of SFA, as described in Chapter 5.5.1. Leave-one-out cross-validation is performed for each sample to predict the best class (lines 18–22). Finally, the best configuration is returned (line 23).

Parallel Execution (Algorithm 6.2): The training of the classifier is well-suited for parallel execution: There is no data-dependency among window lengths in line 7. There are two nested embarrassingly parallel regions in lines 10–11 for each histogram and in lines 18–20 for each 1-NN search.

6.4.2 Computational Complexity

Prediction (Algorithm 6.1): The computational complexity for classification is given by a 1-NN search over the $|CLASSES|$ classes using the Cosine similarity:

$$\begin{aligned} T(BOSS\ VS\ Predict) &\epsilon\ O(|CLASSES| \cdot T(BOSSVSDistance)) \\ &\epsilon\ O(|CLASSES| \cdot n) \end{aligned} \tag{6.9}$$

It has a significantly lower computational complexity than most rivaling approaches (Table 6.1).

Fit (Algorithm 6.2) The computational complexity of the train phase results from (a) building N histograms, one for each of N time series, and (b) building $|CLASSES|$ *tf-idf* vectors, one for each class C. This is done for $\sqrt{n}$ window lengths:

$$\begin{aligned} T(BOSS\ VS\ Fit) &\epsilon\ O(N \cdot \sum_{w=1}^{\sqrt{n}} [T(BOSS\ VS\ Predict) + T(BOSS)]) \\ &\epsilon\ O(Nn^{\frac{3}{2}}\,|CLASSES| + N \cdot \sum_{w=1}^{\sqrt{n}} (n + (\sqrt{n}w)\log(\sqrt{n}w))) \\ &=\ O(Nn^{\frac{3}{2}} + N \cdot \sum_{w=1}^{\sqrt{n}} n \log n) \qquad (6.10) \\ &=\ O(Nn^{\frac{3}{2}} \log n) \qquad (6.11) \end{aligned}$$

6.5 Experiments

6.5.1 Setup

We evaluated the BOSS VS model using two case studies and 91 public time series benchmark datasets (Table 8.1). The web page accompanying the BOSS VS contains all raw numbers and the C++ source codes [73]. The BOSS VS classifier was implemented in C++ using OpenMP and JAVA. All experiments were performed using the JAVA implementation on a shared memory machine running LINUX with 8 Quad Core AMD Opteron 8358 SE processors, and JAVA JDK x64 1.8. For all experiments the time series datasets were z-normalized prior to the experiments (compare Chapter 2.2). All experiments consist of two phases: model building using the train dataset and testing the classification accuracy using the test dataset.

Each dataset provides a *train/test* split. There might be a confusion with the terms "training set", "validation set", and "test set" used in supervised learning. The train split is used as the training and validation set. The test split is used as the test set. By the use of these train/test splits, the results are comparable to previous publications. All our

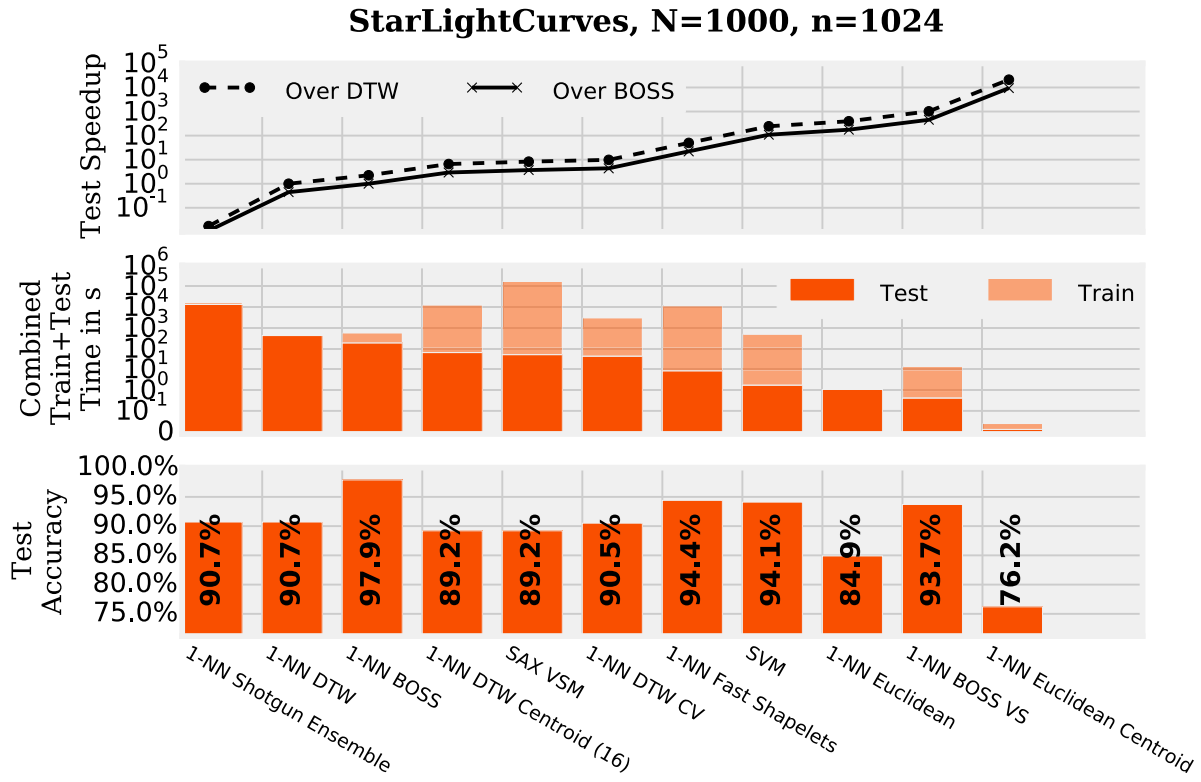


Figure 6.5 – Accuracy and classification times for starlight curves using 32 cores.

results are based on the *test accuracy* of the classifiers using the test split. We use the term *BOSS VS* as an equivalent to the 1-NN BOSS VS classifier.

The 1-NN BOSS VS classifier is compared to state of the art classifiers including nearest-neighbor based classifiers like 1-NN Fast Shapelets [63], 1-NN bag-of-patterns [48], 1-NN Shotgun ensemble [71], 1-NN ED or 1-NN DTW with the optimal warping window constraint [62], SAX-VSM [81], 1-NN BOSS ensemble classifier [68], an ensemble technique *Proportional* (PROP) [49], or machine learning techniques such as support vector machines (SVM) with a quadratic and cubic kernel, and a tree based ensemble method (random forest). Where possible we used the implementations given by the authors, or python using sklearn for the SVM and random forest benchmarks.

6.5.2 Case Studies: Classification

Starlight Curves

The StartLightCurve dataset [42] contains $N = 1000$ train samples and $N = 8236$ test samples each of length 1024. Figure 6.5 illustrates (from top to bottom):

1. The speedup of each classifier over 1-NN DTW (dotted line) and 1-NN BOSS ensemble (straight line),
2. The combined test and train classification times, and
3. The classification accuracy.

Training the classifiers takes from seconds (BOSS VS) up to several days (SAX-VSM, DTW centroid) using 32 cores. We have added a video to our website to illustrate differences in test times [73].

- The 1-NN DTW, using the state of the art implementation [62], takes 7 minutes on our 32 core machine for prediction.
- 1-NN DTW CV uses a warping window constraint set through cross validation and is two orders of magnitude slower than the 1-NN BOSS VS in test time, and it takes roughly one hour to train using all cores.
- The 1-NN BOSS ensemble classifier takes three minutes for the classification task and has the best accuracy score of 97.9%, due to its invariances to noise, phase shifts, offsets, amplitudes and occlusions.
- The Euclidean distance based classifiers are the fastest but show the most inaccurate results.
- Our 1-NN BOSS VS classifier trades off accuracy for improved (reduced) test and train times and takes only 0.4 seconds which is three orders of magnitude faster than 1-NN DTW, and 2.5 orders of magnitude faster than the 1-NN BOSS ensemble classifier.

Personalized Medicine: Heartbeat BIDMC

The BIDMC Congestive Heart Failure Database [3] consists of ECG recordings of 15 subjects, who suffer from severe congestive heart failures. The recordings contain noisy or extraneous data, when the recordings started before the machine was connected to the patient. ECG signals show a high level of redundancy due to repetitive heart beats but even a single patient can have multiple different heart beats. The total size of this dataset is equal to 9 million data points (10 hours sampled at 250 Hz). We used the train/test split provided by [38] containing 600 samples each. The train and test splits have different lengths of 3750 and 11250.

Figure 6.6 shows that the 1-NN BOSS ensemble, the 1-NN BOSS VS and the 1-NN Shotgun Ensemble classifiers offer the best classification accuracy. The other classifiers have a much lower classification accuracy. This is not surprising as the data is noisy and requires invariance to phase shifts in order to cope with the periodic ECG patterns. Train times vary from minutes (BOSS, BOSS VS) to days (SAX-VSM, Shotgun classifier, and

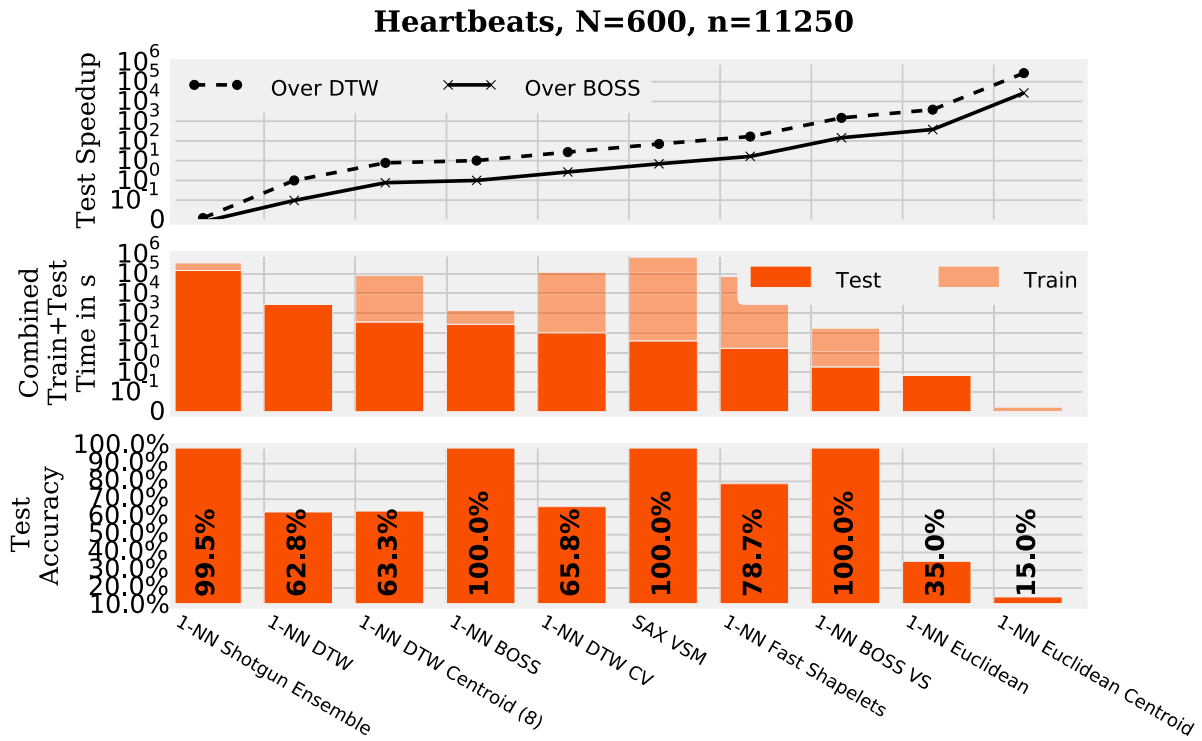


Figure 6.6 – Accuracy and classification times for ECG recordings using 32 cores. SVM was not benchmarked due to variable length time series. Fast Shapelets timed out after 6 days of training.

DTW centroid) using 32 cores. 1-NN DTW has the highest test time with 30 minutes. The 1-NN Euclidean Centroid classifier has the lowest test time with 0.01s, but the lowest accuracy.

Our 1-NN BOSS VS classifier offers the best trade-off between train/test times and accuracy. The test time is three orders of magnitude lower than that of 1-NN DTW and two orders of magnitude lower than the 1-NN BOSS ensemble classifier. Even when combining the test time of 2s and the train time of 150s, the 1-NN BOSS VS is one order of magnitude faster than the 1700s 1-NN DTW classifier's test time.

Implications: The 1-NN BOSS VS classifier provides high speed with good accuracy, which makes it unique and relevant for many practical use cases.

6.5.3 Classification Accuracy Benchmark

All classifiers were evaluated using the same 91 public time series datasets (see Appendix Table 8.1):

- 45 datasets come from the UCR time series classification archive [42].

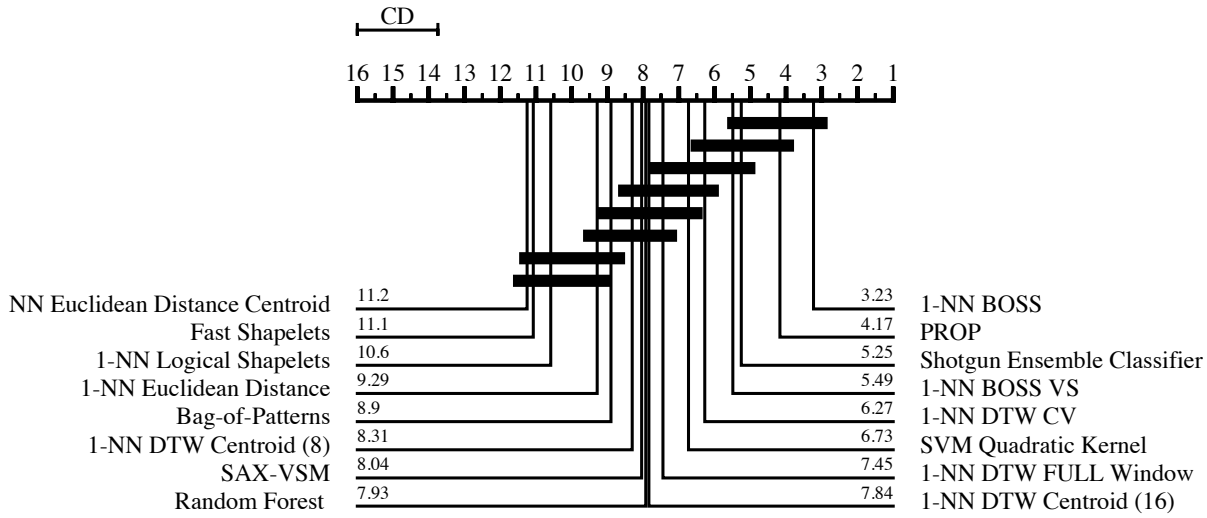


Figure 6.7 – Critical difference diagram for state of the art classifiers on the 91 datasets (Appendix Table 8.1). The best classifiers are to the right. Critical difference (CD) is 2.2.

- 46 datasets come from other publications [12, 14, 29, 48, 52, 63, 81, 42].

Please see Chapter 6.5.1 regarding the use of the *train/test* splits for testing and training.

Classification Accuracy

Figure 6.7 shows a critical difference diagram over the average ranks of the classifiers as introduced in [28]. The classifiers with the lowest (best) ranks are to the right. The group of classifiers that are not significantly different in their rankings are connected by a bar. The critical difference (CD) length is shown above the graph.

- The 1-NN BOSS ensemble classifier performs best and on average shows the lowest rank. This confirms our claims that alignment-free similarity in combination with noise reduction is important for time series similarity.

- The PROP [49] ensemble is the second best classifier. It is an ensemble of 11 classifiers including 1-NN DTW CV, 1-NN DTW, 1-NN LCSS, 1-NN ED, etc. As such it has an enormous classification time, as to predict a label it has to run all classifiers first.

- Our Shotgun Ensemble classifier is the third best classifier. It provides alignment-free similarity but lacks tolerance to noise.

- The strength of the 1-NN BOSS VS classifier lies in its computational complexity rather than its high classification accuracy. Still its accuracy is not significantly lower than that of the best classifiers 1-NN BOSS, PROP, 1-NN Shotgun ensemble, or 1-NN DTW CV.

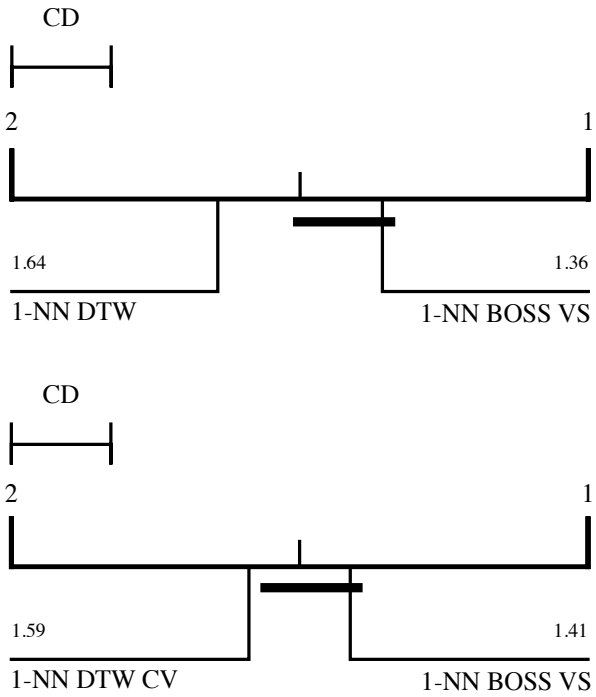


Figure 6.8 – Critical difference diagram for BOSS VS vs 1-NN DTW with full window (top) and 1-NN DTW CV with the warping window constraint set through cross validation (bottom) on the 91 datasets (Appendix Table 8.1). Critical difference is 0.17 for both.

- The 1-NN DTW with a full warping window is commonly used as the benchmark to compare to [29, 13, 49] and performs worse than all of the three similarity models presented in this thesis. The same applies for the 1-NN DTW CV classifier.
- The other classifiers perform significantly worse, including the SAX-VSM classifier that builds on SAX and the vector space model, or Shapelet classifiers.

Implications: This experiment underlines the importance of alignment-free similarity (Shotgun ensemble distance) and noise tolerance (BOSS model) for the time series data analytics task. The similarity models presented in this thesis score three out of the first four ranks on the 91 public time series datasets. The 1-NN BOSS ensemble is the most accurate classifier up to this day and the 1-NN BOSS VS classifier is extremely fast and provides good accuracy, which is better than the benchmark 1-NN DTW classifier.

Is 1-NN BOSS VS significantly more accurate than 1-NN DTW with a full warping window?

A central claim is that "a new algorithm is only of interest in terms of accuracy if it can significantly outperform 1-NN DTW with a full warping window" [13]. Among the three classifiers presented in this thesis, the 1-NN BOSS VS classifier has the lowest accuracy.

Thus, we test if this classifier performs significantly better than the 1-NN DTW with or without a warping window constraint using all 91 time series datasets (Figure 6.8).

Critical difference diagram: Figure 6.8 shows the critical difference diagrams. The critical difference is 0.17 in both cases. The 1-NN BOSS VS is significantly more accurate than 1-NN DTW with a difference in ranks of $1.64 - 1.36 > 0.17$ and $1.59 - 1.41 > 0.17$ and:

- 1-NN BOSS VS has 57 wins, 3 draw, and 31 losses over 1-NN DTW with a full warping window.
- 1-NN BOSS VS has 53 wins, 1 draw, 37 losses over 1-NN DTW CV (see the excel sheet with detailed results [73]).

Wilcoxon signed rank test: To validate the results, we performed a Wilcoxon signed rank test to check if 1-NN BOSS VS is significantly different from 1-NN DTW FULL and 1-NN DTW CV. With a p-value of 0.03246 for 1-NN DTW VS and 0.0001703 for 1-NN DTW FULL we can reject the null-hypothesis that the two 1-NN DTW classifiers are from the same distribution in both cases.

Implications: As the 1-NN BOSS VS classifier is significantly more accurate than 1-NN DTW, the same applies to the 1-NN BOSS ensemble and 1-NN Shotgun ensemble classifiers.

Pairwise comparison of wall-clock times

Figure 6.9 shows the wall-clock times of the four state of the art classifiers in a pairwise comparison to 1-NN BOSS VS. PROP is an ensemble of classifiers including 1-NN DTW and as such cannot be faster than 1-NN DTW or 1-NN DTW CV. Again our BOSS VS classifier significantly outperforms the other classifiers by up to four orders of magnitude in terms of test times.

- The 1-NN BOSS VS classifier is significantly faster than the 1-NN BOSS ensemble classifier in terms of train and test times.
- It seems to have similar training times as the 1-NN Shotgun ensemble classifier. When looking at the raw data, the 1-NN Shotgun ensemble classifier trains faster for very small datasets and is orders of magnitude slower for moderate to large sized datasets.
- 1-NN DTW CV requires a training phase, resulting in reduced test times when compared to 1-NN DTW. When looking at the raw data, the 1-NN DTW CV classifier has similar test times for the small datasets and is orders of magnitude

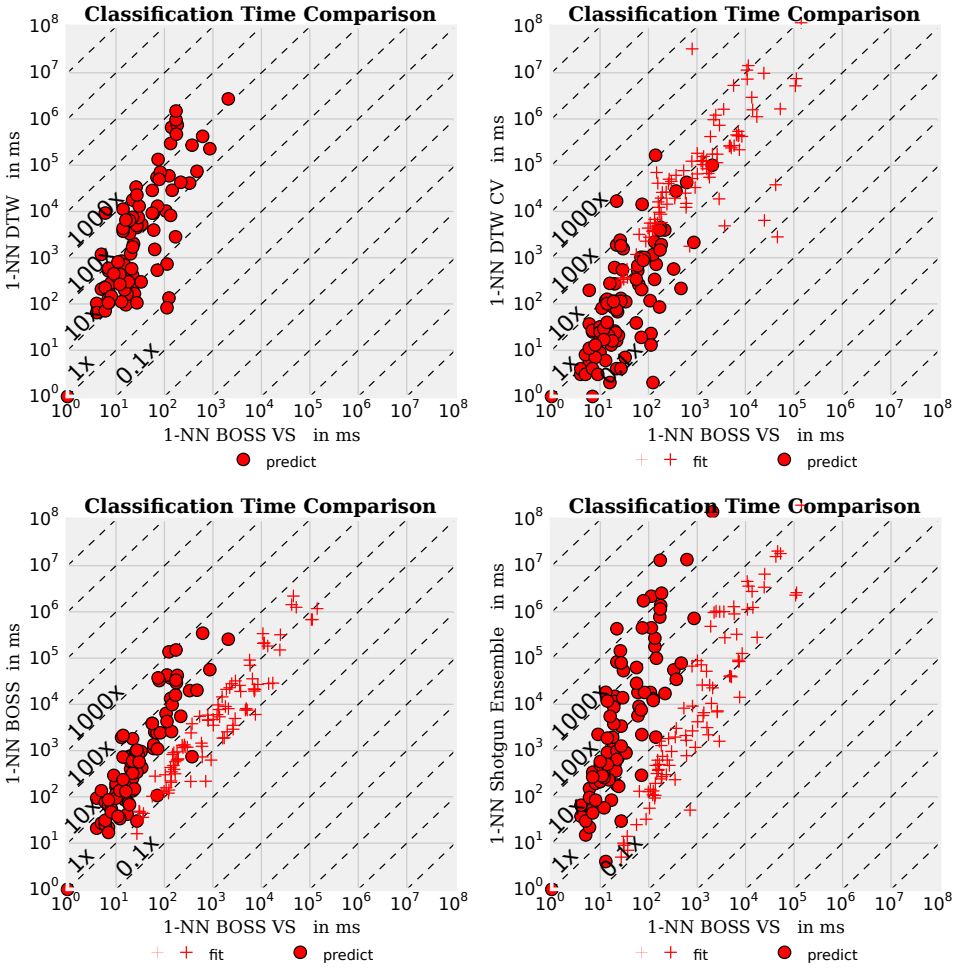


Figure 6.9 – Test and train wall-clock times of the four most accurate classification algorithms compared to 1-NN BOSS VS on the 91 datasets (Appendix Table 8.1).

slower for moderate to large sized datasets (compare total time to completion in Figure 6.1).

- 1-NN DTW is one to four orders of magnitude slower in terms of test times than our 1-NN BOSS VS, and is never significantly faster.

Implications:

The 1-NN BOSS VS is not the most accurate classifier, but it is (a) multiple orders of magnitude faster than the 1-NN BOSS ensemble classifier, 1-NN DTW, 1-NN Shotgun ensemble classifier, and state of the art, and (b) it is significantly more accurate than 1-NN DTW with or without a warping window constraint.

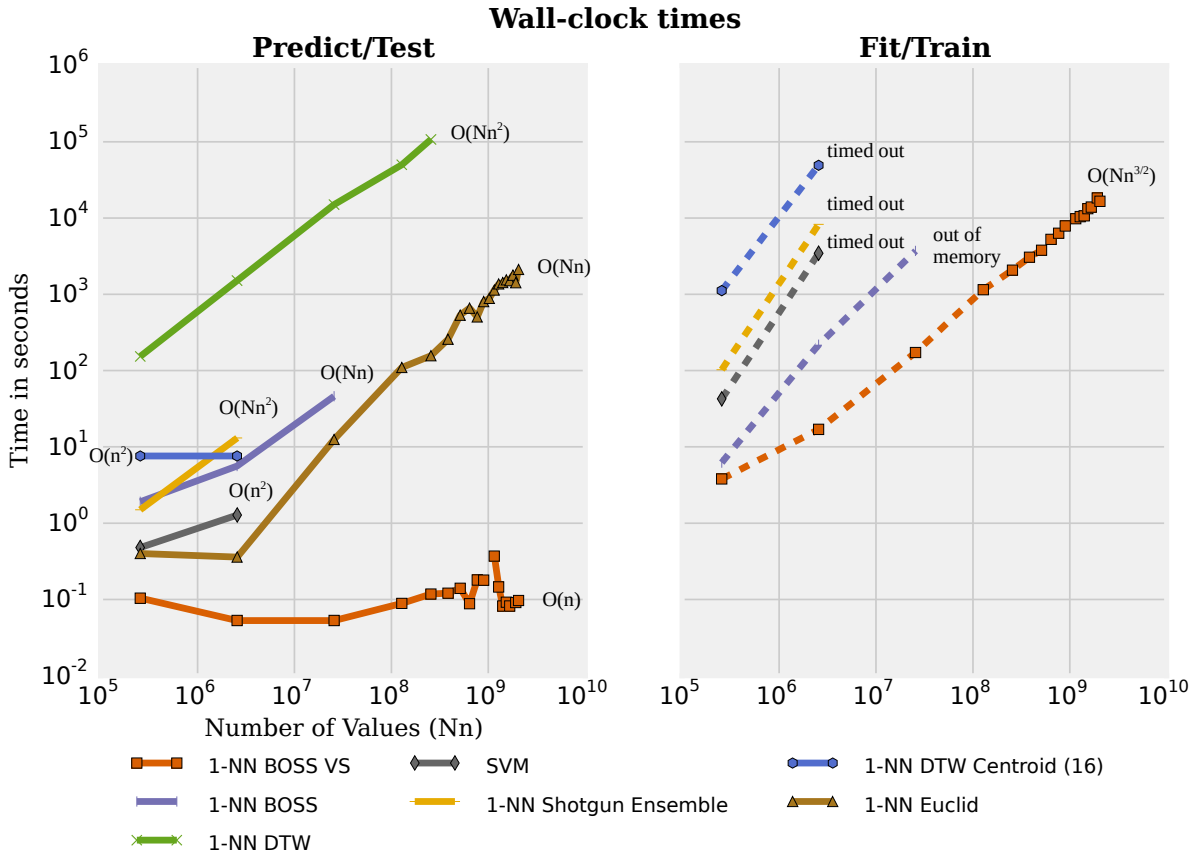


Figure 6.10 – Scalability for an increasing number of time series. Dotted lines represent the train time, solid lines represent the test times.

6.5.4 Scaling to a Billion Values

This is a synthetic benchmark to show the scalability in terms of wall-lock times. We test the scalability based on the Cylinder-Bell-Funnel (CBF) dataset using a variable number of time series N of 10^3 up to $8 \cdot 10^6$ and a fixed time series length $n = 256$ (Figure 6.10). The largest tested dataset has 2.048 billion values ($= 2 \cdot 10^9$) which is roughly equal to 15 GB of data. This synthetic benchmark dataset is widely used and contains three basic shapes: Cylinders, Bells and Funnels. We performed 3000 predictions for each classifier. We stopped a classifier when it took more than several days to train. This happened for 1-NN DTW Centroid, SVMs and the 1-NN Shotgun Ensemble. All experiments were performed using 32 cores.

We focus on execution time rather than accuracy in this experiment. Figure 6.10 shows two sets of curves for training (left) and testing (right) the classifiers.

Training the 1-NN BOSS VS classifier took a maximum of 4.4 hours for the largest dataset size of two billion values. These train times are quickly amortized given the

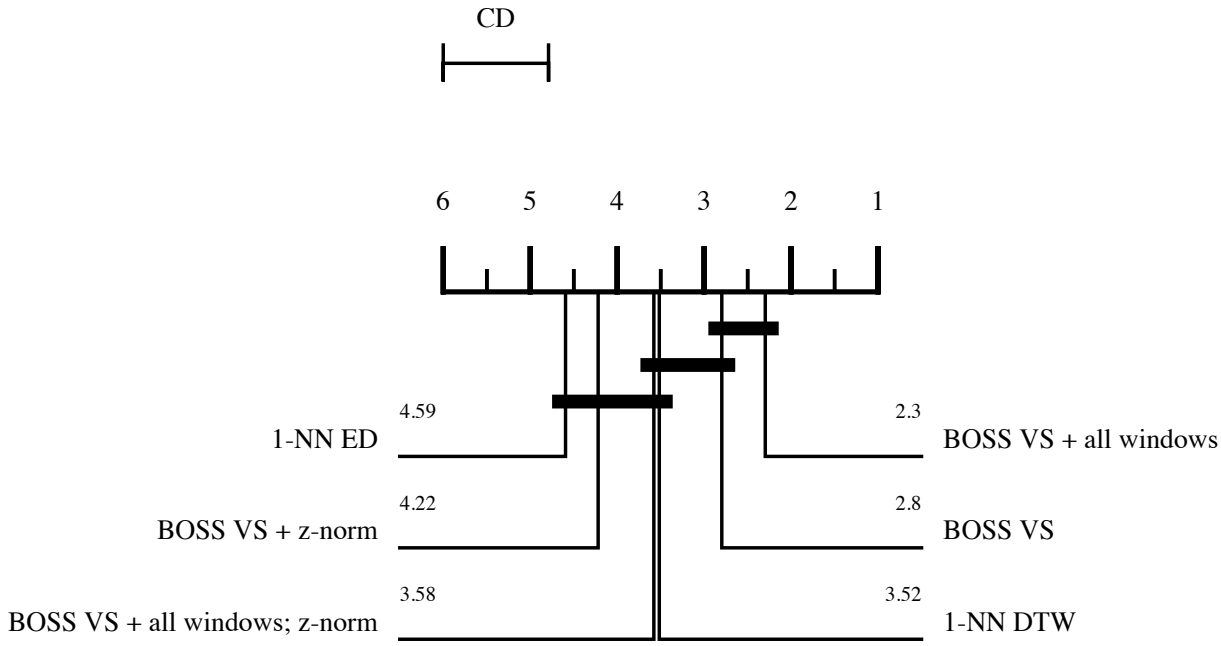


Figure 6.11 – Critical difference diagram for the design decisions made using the 45 UCR datasets. The best classifiers are to the right. Critical difference (CD) is 1.2.

presented 1-NN DTW prediction times. 1-NN DTW took 27 hours to predict 0.2 billion values, where 1-NN BOSS VS takes less than 1/50 of the time to train. We didn't continue to measure the 1-NN DTW times from there on.

1-NN BOSS VS testing took on average 10^{-1} seconds for all 3000 predictions regardless of the dataset size. The other classifiers scale with the size of the dataset, with the exception of 1-NN DTW centroid which has an enormous train time and timed out after 7 days for datasets larger than $2 \cdot 10^6$. The 1-NN BOSS ensemble runs out of memory at $2 \cdot 10^7$ values.

Implications: In total the 1-NN BOSS VS classifier is multiple orders of magnitude faster than the other classifiers and has moderate train times. These empirical results are not surprising given the test complexities of the other classifiers and the fact that our 1-NN BOSS VS has a complexity of $O(n)$ (compare Table 6.1).

6.5.5 Impact of Design Decisions

We use all 46 UCR datasets [42] to test the impact of our design decisions. The BOSS VS is based on four design decisions:

1. Testing a *subset* of $\sqrt{n}$ windows for training as opposed to using *all windows (BOSS VS + all windows*).

2. *Mean normalization* as a parameter as opposed to always applying z-normalization *(BOSS VS + z-norm)* to all windows.

The BOSS VS was designed to use a subset of windows and mean normalization as a parameter. Figure 6.11 shows that the mean normalization as a parameter performs significantly better than always norming the data (*"+z-norm"*). The use of $\sqrt{n}$ windows performs worse than the use of all windows (*"+all windows"*).

Implications: The BOSS VS model is based on using a subset of windows for training and mean normalization as a parameter. While the mean normalization improves the accuracy, the use of a subset of windows reduces accuracy. However, it is crucial for a low train time, which is reduced to $O(Nn^{\frac{3}{2}})$.

6.6 Summary

In the context of mining large datasets and real-time analytics there is a need for time series classification algorithms with (a) a low test time to allow for mining large datasets, (b) a moderate train time to allow for frequent model updates, (c) tolerance to noise to provide high classification accuracy, and (d) offering alignment-free similarity.

Our BOSS in Vector Space (BOSS VS) model builds on the BOSS model for alignment-free and noise-robust time series data analytics combined with the vector space model. It extends the BOSS model by a compact representation of classes instead of time series, which significantly reduces the computational complexity, weights characteristic SFA words, but trades off accuracy for improved train and test times. In the vector space model the term frequency - inverse document frequency model is used. The approach makes the BOSS VS model alignment-free, robust to noise and scalable in execution times.

The 1-NN BOSS VS has a low test complexity which allows for the classification of massive time series datasets. Its moderate train complexity, which is lower than the test complexity of 1-NN DTW, allows for frequent model updates as in real-time predictive analytics.

As part of our experimental evaluation, we show that the 1-NN BOSS VS classifier scales to up to two billion values (15 GB), and it is multiple orders of magnitude faster than state of the art with a similar level of classification accuracy. The 1-NN BOSS VS classifier is the fourth best classifier with regards to classification accuracy. Our 1-NN BOSS ensemble classifier (Chapter 5) shows the best classification accuracy, followed by an ensemble technique PROP, and the 1-NN Shotgun Ensemble classifier (Chapter 4).

The 1-NN BOSS VS is not the most accurate classifier. However, its high speed (up to four orders of magnitude faster) combined with its good classification accuracy make the BOSS VS model unique and relevant for many practical use cases.

The BOSS VS model aims at classification. Other analytics task such as motif discovery and discord discovery are easily applicable.

Chapter 7

Conclusion

7.1 Summary and Results

Working with time series is difficult due to the high dimensionality of the data, erroneous or extraneous data, and large datasets. At the core of time series data analytics there are (a) a *time series representati*on and (b) a *similarity measure* to compare two time series. There are many desirable properties of similarity measures[1]. Common similarity measures in the context of time series are Dynamic Time Warping (DTW) or the Euclidean Distance (ED). However, these are decades old and do not meet today's requirements. We believe that the over-dependance of research on the UCR time series classification benchmark has led to two pitfalls, namely: (a) they focus mostly on accuracy[2] and (b) they assume pre-processed datasets[3]. We identified three (additional) desirable properties: (a) alignment-free structural similarity, (b) noise-robustness, and (c) scalability.

This dissertation aims at the core of time series data analytics by introducing a symbolic time series representation and three time series similarity measures for alignment-free, noise-robust and scalable time series data analytics. In this context, the contributions of this dissertation are as follows:

[1] "A similarity measure should be consistent with our intuition and provide the following properties. (1) It should provide a recognition of perceptually similar objects, even though they are not mathematically identical. (2) It should be consistent with human intuition. (3) It should emphasize the most salient features on both local and global scales. (4) A similarity measure should be universal in the sense that it allows to identify or distinguish arbitrary objects, that is, no restrictions on time series are assumed. (5) It should abstract from distortions and be invariant to a set of transformations."[30]

[2] "There are numerous metrics to assess new algorithms for TSC [Time Series Classification]. However, accuracy is, in our opinion, the most important."[49]

[3] "Moreover, in virtually all time series classification research, the data must be arranged to have equal length. For example, in the world's largest collection of time series datasets, the UCR classification archive, all forty-five time series datasets contain only equal-length data."[38]

A Symbolic Time Series Representation: The symbolic time series representation Symbolic Fourier Approximation (SFA) represents each real-valued time series by a string. SFA is composed of approximation using the Fourier transform and quantization using a technique called Multiple Coefficient Binning (MCB). Among its properties, the most notable are: (a) noise removal due to low-pass filtering and quantization, (b) the string representation due to quantization, and (c) the frequency domain nature of the Fourier transform. The frequency domain nature makes SFA unique among the symbolic time series representations. Dynamically adding or removing Fourier coefficients to adapt the degree of approximation is at the core of the algorithms presented in this thesis.

Three Similarity Measures: Over the past decade similarity measures were designed mostly to work with human assistance to extract characteristic patterns and align and trim them for equivalent-length and scaling (aka the alignment). A similarity measure is alignment-free if it provides the alignment of characteristic patterns. This dissertation strives towards alignment-free, scalable time series analytics in the presence of noise by introducing three novel time series similarity models:

- Our *Shotgun distance* model is based on the structural similarity of time series and thereby reduces the need for cost-ineffective pre-processing for alignment. The Shotgun distance model breaks a query time series into subsequences and vertically aligns and horizontally scales each subsequence prior to measuring the similarity to a sample time series. This reduces the need for human pre-processing of a time series in order to extract equivalent-length and aligned characteristic subsequences (patterns).

- Our *Bag-Of-SFA-Symbols (BOSS)* model combines the noise tolerance of the time series representation Symbolic Fourier Approximation (SFA) with the structure-based representation of the bag-of-words model which makes it inherently alignment-free. Apart from invariance to noise, the BOSS model provides invariances (robustness) to phase shifts, offsets, amplitudes and occlusions by discarding the original ordering of the SFA words and normalization. This leads to the highest classification and clustering accuracy in time series literature to date.

- Our *BOSS in Vector Space (BOSS VS)* model builds upon the BOSS model for alignment-free and noise-robust time series data analytics combined with the vector space model (term frequency-inverse document frequency model). It significantly reduces the computational complexity of the BOSS model to allow for the classification of massive time series datasets. Its moderate train complexity, which is lower than the test complexity of 1-NN DTW, allows for frequent model updates such as mining streaming data (aka real-time predictive analytics). The 1-NN BOSS VS is not the most accurate classifier. However, its high speed combined with its good accuracy makes it unique and relevant for many practical use cases.

Time Series Data Analytics Tasks: We presented algorithms and use cases in the context of:

- *Dimensionality Reduction*: We have introduced the symbolic representation SFA. SFA performs significantly better than many other dimensionality reduction techniques including those techniques based on mean values like SAX, PLA, PAA, or APCA. This is due the fact, that SFA builds upon DFT, which is significantly more accurate than the other dimensionality reduction techniques.

- *Indexing/Similarity Search*: We have introduced our SFA trie for time series similarity search. Our SFA trie is the best spatial access method in terms of page accesses and wall-clock time on real and synthetic datasets, allows for indexing terabyte-sized time series datasets in main memory, and performs better than state of the art, like the iSAX 2.0 index, the TS-tree, KD-tree, or R*-tree.

- *Classification and Accuracy*: Our similarity models represent three of the four most accurate classifiers to date using 91 public time series benchmark datasets. Our 1-NN BOSS ensemble classifier shows the highest classification accuracy among 16 techniques, followed by an ensemble technique PROP [13, 49], our 1-NN Shotgun Ensemble classifier, and our 1-NN BOSS VS classifier. This is mainly due to the alignment-free or noise-robust properties of our similarity measures.

- *Classification and Scalability:* The 1-NN BOSS VS classifier is one to four orders of magnitude faster than state of the art and significantly more accurate than the 1-NN DTW classifier, which serves as the benchmark to compare to[4] [5]. I.e., we can solve a classification problem with 1-NN DTW CV that runs on a cluster of 4000 cores for one day [13], with the 1-NN BOSS VS classifier using commodity hardware and a 4 core cpu within one to two days resulting in a similar or better classification accuracy.

- *Clustering*: We presented use cases for unaligned time series datasets, which underline that our similarity models provide a higher clustering accuracy than state of the art (1-NN DTW and 1-NN ED), due to the alignment-free similarity and noise-robustness.

- *Computational Bioacoustics*: We showed that the Shotgun distance in combination with SFA is applicable to computational bioacoustics and performs better than state of the art feature extraction techniques in the context of audio processing.

[4] "A new algorithm is only of interest in terms of accuracy if it can significantly outperform 1-NN DTW with a full warping window."[13]

[5] "We think that a new algorithm for TSC [Time Series Classification] is only of interest to the data mining community if it can significantly outperform DTWCV."[49]

7.2 Future Directions

Extract Relevant Frequencies: For SFA we decided to use the lowest Fourier coefficients, which serves as a low-pass filter. However, the approach is not limited to the use of the lowest coefficients. When processing audio data, the relevant data is typically present in a specific frequency band. Currently there is no method that automatically extracts relevant frequencies. Instead, domain knowledge can be used to extract the corresponding Fourier coefficients. In fact, this was done for classifying flying insects (Chapter 4.6.4).

Parameter-free Data Mining: Our similarity measures have a set of two to four parameters. These parameters depend on the dataset and data analytics method, and can not be set implicitly. Training is required to fine-tune the parameters. It would be desirable to have parameter-free methods such as for clustering.

Feature Matching: The presented classifiers are based on 1-NN classification. Our models are not limited to this classification approach. Other feature matching techniques such as SVMs, decision trees, random forests, etc., could be applied, too.

Data Analytics Tasks: Other typical time series data analytics tasks include motif discovery, or anomaly detection (discord discovery). It is straightforward to use the term frequencies of the SFA words in the BOSS VS model to find low frequency (unusual) or high frequency (frequent) SFA words. This could be a starting point for motif discovery or discord discovery.

Chapter 8

Appendix

45 UCR datasets [42]			
50words	ECGFiveDays	MALLAT	Symbols
Adiac	FaceAll	MedicalImages	synthetic_control
Beef	FaceFour	Motes	Trace
CBF	FacesUCR	NonInvasiveFatalECG_Thorax1	Two_Patterns
ChlorineConcentration	Fish	NonInvasiveFatalECG_Thorax2	TwoLeadECG
CinC_ECG_torso	Gun-Point	OliveOil	uWaveGestureLibrary_X
Coffee	Haptics	OSULeaf	uWaveGestureLibrary_Y
Cricket_X	InlineSkate	SonyAIBORobotSurface	uWaveGestureLibrary_Z
Cricket_Y	ItalyPowerDemand	SonyAIBORobotSurfaceII	wafer
Cricket_Z	Lighting2	StarlightCurves	WordsSynonyms
DiatomSizeReduction	Lighting7	SwedishLeaf	yoga
ECG200			
46 Datasets from [12, 14, 29, 48, 52, 63, 81, 42, 5]			
ArrowHead	FordB	Passgraph	stig
ARSim	Ham	PhalangesOutlinesCorrect	Strawberry
BeetleFly	HandOutlines	Phoneme	ToeSegmentation1
BirdChicken	heartbeat (BIDMC)	ProximalPhalanxOutlineAgeGroup	ToeSegmentation2
Computers	Herring	ProximalPhalanxOutlineCorrect	UWaveGestureLibraryAll
DistalPhalanxOutlineAgeGroup	InsectWingbeatSound	ProximalPhalanxTW	wheat
DistalPhalanxOutlineCorrect	LargeKitchenAppliances	RefrigerationDevices	Wine
DistalPhalanxTW	Meat	ScreenType	WordSynonyms
Earthquakes	MiddlePhalanxOutlineAgeGroup	ShapeletSim	Worms
ECG5000	MiddlePhalanxOutlineCorrect	ShapesAll	WormsTwoClass
ElectricDevices	MiddlePhalanxTW	shield	
FordA	Otoliths	SmallKitchenAppliances	

Table 8.1 – The 91 time series datasets used in the classification/clustering experiments.

Bibliography

[1] UCR Insect Contest. http://www.cs.ucr.edu/~eamonn/CE, 2012.

[2] Kaggle: Go from Big Data to Big Analytics. https://www.kaggle.com, 2014.

[3] The BIDMC congestive heart failure database. http://www.physionet.org/physiobank/database/chfdb/, 2014.

[4] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Foundations of Data Organization and Algorithms*, 1993.

[5] Osama Al-Jowder, Marianne Defernez, E Katherine Kemsley, and Reginald H Wilson. Mid-infrared spectroscopy and chemometrics for the authentication of meat products. *Journal of agricultural and food chemistry*, 47(8):3210–3218, 1999.

[6] S Albrecht, I Cumming, and J Dudas. The momentary fourier transformation derived from recursive matrix transformations. In *Digital Signal Processing Proceedings, 1997.* IEEE, 1997.

[7] Tarek Amr. Survey on Time-Series Data Classification. In *TSDM 2012*, 2012.

[8] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[9] Cláudia M Antunes and Arlindo L Oliveira. Temporal data mining: An overview. In *KDD workshop*, pages 1–13, 2001.

[10] Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The TS-tree: efficient time series search and retrieval. In *EDBT*, pages 252–263, New York, NY, USA, 2008. ACM.

[11] Jean-Julien Aucouturier, Boris Defreville, and Francois Pachet. The bag-of-frames approach to audio pattern recognition: A sufficient model for urban soundscapes but not for polyphonic music. *The Journal of the Acoustical Society of America*, 122(2):881–891, 2007.

[12] Anthony Bagnall, Luke M. Davis, Jon Hills, and Jason Lines. Transformation Based Ensembles for Time Series Classification. In *SDM*. SIAM / Omnipress, 2012.

[13] Anthony Bagnall and Jason Lines. An Experimental Evaluation of Nearest Neighbour Time Series Classification. *arXiv preprint arXiv:1406.4757*, 2014.

[14] Gustavo Batista, Xiaoyue Wang, and Eamonn J. Keogh. A Complexity-Invariant Distance Measure for Time Series. In *SDM*. SIAM / Omnipress, 2011.

[15] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660. ACM, 2005.

[16] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD*, 19(2):322–331, 1990.

[17] Richard Bellman and Robert Kalaba. On adaptive control processes. *Automatic Control, IRE Transactions on*, 4(2):1–9, 1959.

[18] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[19] Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *SIGMOD*, pages 599–610. ACM, 2004.

[20] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. iSAX 2.0: Indexing and Mining One Billion Time Series. *ICDM*, 0:58–67, 2010.

[21] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *KAIS*, 39(1):123–151, 2014.

[22] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *SIGMOD*, pages 151–162, 2002.

[23] Franky Kin-Pong Chan, Ada Wai chee Fu, and Clement Yu. Haar Wavelets for Efficient Similarity Search of Time-Series: With and Without Time Warping. *TKDE*, 15(3):686–705, 2003.

[24] Kin-Pong Chan and Ada Wai-Chee Fu. Efficient Time Series Matching by Wavelets. In *ICDE*, pages 126–133, 1999.

[25] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. Indexable PLA for Efficient Similarity Search. In *VLDB*. ACM, 2007.

[26] CMU Graphics Lab Motion Capture Database. CMU Graphics Lab Motion Capture Database. `http://mocap.cs.cmu.edu/`, 2014.

[27] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Computational Geometry*, pages 253–262. ACM, 2004.

[28] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[29] Hui Ding. Querying and mining of time series data: experimental comparison of representations and distance measures. *VLDB*, 2008.

[30] Philippe Esling and Carlos Agon. Time-series data mining. *CSUR*, 45(1):12, 2012.

[31] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[32] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[33] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[34] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[35] Manish Gupta, Jing Gao, Charu Aggarwal, and Jiawei Han. Outlier detection for temporal data. *DMKD*, 5(1):1–129, 2014.

[36] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57. ACM, 1984.

[37] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[38] Bing Hu, Yanping Chen, and Eamonn Keogh. Time Series Classification under More Realistic Assumptions. In *SDM*, 2013, 2013.

[39] Zbigniew Jerzak and Holger Ziekow. The DEBS 2014 grand challenge. In *DEBS*, pages 266–269. ACM, 2014.

[40] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 2001.

[41] E. Keogh and M. Pazzani. A Simple Dimensionality Reduction Technique for Fast Similarity Search in Large Time Series Databases. In *PAKDD*, volume 1805, pages 122–133. Springer, 2000.

[42] E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. UCR Time Series Classification/Clustering Homepage. `http://www.cs.ucr.edu/~eamonn/time_series_data`, 2014.

[43] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *KDD*, pages 102–111. ACM, 2002.

[44] Flip Korn, H. V. Jagadish, and Christos Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD*, pages 289–300. ACM, 1997.

[45] Nitin Kumar, Venkata Nishanth Lolla, Eamonn J. Keogh, Stefano Lonardi, and Chotirat (Ann) Ratanamahatana. Time-series Bitmaps: a Practical Visualization Tool for Working with Large Time Series Databases. In *SDM*, 2005.

[46] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, pages 2–11, 2003.

[47] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *DMKD*, 2007.

[48] Jessica Lin, Rohan Khade, and Yuan Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *JIIS*, 2012.

[49] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *DMKD*, pages 1–28, 2014.

[50] Constanze Lipowsky, Egor Dranischnikow, Herbert Göttler, et al. Alignment of Noisy and Uniformly Scaled Time Series. In *DEXA*, Lecture Notes in Computer Science. Springer, 2009.

[51] N. Marwan, M. Thiel, and N. R. Nowaczyk. Cross recurrence plot based synchronization of time series. *Nonlinear Processes in Geophysics*, 9(3/4):325–331, 2002.

[52] Abdullah Mueen, Eamonn J. Keogh, and Neal Young. Logical-shapelets: an expressive primitive for time series classification. In *KDD*. ACM, 2011.

[53] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. The DEBS 2013 grand challenge. In *DEBS*, pages 289–294. ACM, 2013.

[54] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online Amnesic Approximation of Streaming Time Series. In *ICDE*, pages 338–349, 2004.

[55] Petra Perner. *Data mining on multimedia data*, volume 2558. Springer, 2002.

[56] François Petitjean, Germain Forestier, Geoffrey I. Webb, et al. Dynamic Time Warping Averaging of Time Series allows Faster and more Accurate Classification. In *ICDM*, 2014.

[57] Ivan Popivanov. Similarity search over time series data using wavelets. In *ICDE*, pages 212–221, 2002.

[58] Ilyas Potamitis and Patrick Schäfer. On Classifying Insects from their Wing-Beat: New Results. *Ecology and acoustics: emergent properties from community to landscape, Paris, France*, 2014.

[59] Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 26:853–854, March 2010.

[60] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, pages 13–25, New York, NY, USA, 1997. ACM.

[61] D. Rafiei and A. Mendelzon. Efficient Retrieval of Similar Time Sequences Using DFT. In *FODO*, pages 249–257, 1998.

[62] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, et al. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*. ACM, 2012.

[63] Thanawin Rakthanmanon and Eamonn Keogh. Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets. In *SDM*, 2013.

[64] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, New York, NY, USA, 1995. ACM.

[65] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *ICASSP*, pages 43–49, 1978.

[66] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[67] P. Schäfer. Zeitreihen in Datenbanksystemen. Master's thesis, Free University Berlin, 2008.

[68] P. Schäfer. The BOSS is concerned with time series classification in the presence of noise. *DMKD*, 2014.

[69] P. Schäfer. Experiencing the Shotgun Distance for Time Series Analysis. *Transactions on Machine Learning and Data Mining*, 7(1):3 – 25, 2014.

[70] P. Schäfer. Shotgun Distance Webpage. `http://www.zib.de/patrick.schaefer/shotgun`, 2014.

[71] P. Schäfer. Towards Time Series Classification without Human Preprocessing. In *MLDM*, pages 228–242. Springer, 2014.

[72] P. Schäfer. Webpage, The BOSS. `http://www.zib.de/patrick.schaefer/boss/`, 2014.

[73] P. Schäfer. Webpage, The BOSS in Vector Space. `http://www.zib.de/patrick.schaefer/bossVS/`, 2014.

[74] P. Schäfer. Scalable Time Series Classification. *DMKD*, Accepted.

[75] P. Schäfer and S. Dreßler. Shooting Audio Recordings of Insects with SFA. In *AmiBio Workshop, Bonn, Germany*, 2013.

[76] P. Schäfer and M. Högqvist. SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *EDBT*. ACM, 2012.

[77] P. Schäfer and M. Högqvist. SFA web page. `http://www.zib.de/patrick.schaefer/sfa/`, 2014.

[78] Thomas Seidl and Hans-Peter Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In *SIGMOD*, pages 154–165, 1998.

[79] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *VLDB*, pages 507–518, 1987.

[80] Pavel Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, pages 1–23, 2008.

[81] Pavel Senin and Sergey Malinchik. SAX-VSM: Interpretable Time Series Classification Using SAX and Vector Space Model. In *ICDM*, pages 1175–1180. IEEE, 2013.

[82] J. Shieh and E. Keogh. iSAX: indexing and mining terabyte sized time series. In *KDD*, pages 623–631, 2008.

[83] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse kNN search in arbitrary dimensionality. In *VLDB*, pages 744–755. VLDB Endowment, 2004.

[84] Yufei Tao, Man Lung Yiu, and Nikos Mamoulis. Reverse nearest neighbor search in metric spaces. *TKDE*, 18(9):1239–1252, 2006.

[85] Jürgen Urbanski and Matthias Weber. Big Data im Praxiseinsatz–Szenarien, Beispiele, Effekte. `http://www.bitkom.org/files/documents/BITKOM_LF_big_data_2012_online(1).pdf`, 2012.

[86] J. C. Venter and Others. The Sequence of the Human Genome. *Science*, 291(5507):1304–1351, 2001.

[87] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, 2002.

[88] T. Warren Liao. Clustering of time series data—a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[89] Congjun Yang and King-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, pages 485–492. IEEE, 2001.

[90] Lexiang Ye and Eamonn J. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*. ACM, 2009.

[91] Lexiang Ye and Eamonn J. Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *DMKD*, 2011.

[92] Jesin Zakaria, Abdullah Mueen, and Eamonn J. Keogh. Clustering Time Series Using Unsupervised-Shapelets. In *ICDM*. IEEE Computer Society, 2012.

[93] Shichao Zhang, Chengqi Zhang, and Qiang Yang. Data Preparation for Data Mining. *Applied Artificial Intelligence*, 17(5-6):375–381, 2003.

[94] Seyedjamal Zolhavarieh, Saeed Aghabozorgi, and Ying Wah Teh. A Review of Subsequence Time Series Clustering. *The Scientific World Journal*, 2014, 2014.

Publications of the Author

[Birkenheuer et al., 2010] Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., d. Santos Vieira, I., Fels, G., Gesing, S., Grunzke, R., Herres-Pawlis, S., Kohlbacher, O., Kruber, N., Krüger, J., Lang, U., Packschies, L., Müller-Pfefferkorn, R., Schäfer, P., Schmalz, H.-G., Steinke, T., Warzecha, K.-D., and Wewior, M. (2010). Mosgrid – a molecular simulation grid as a new tool in computational chemistry, biology and material science.

[Birkenheuer et al., 2011a] Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., dos Santos Vieira, I., Fels, G., Gesing, S., Grunzke, R., Herres-Pawlis, S., Kohlbacher, O., Krüger, J., Lang, U., Packschies, L., Müller-Pfefferkorn, R., Schäfer, P., Schmalz, H.-G., Steinke, T., Warzecha, K., and Wewior, M. (2011a). A molecular simulation grid as new tool for computational chemistry, biology and material science. *Journal of Cheminformatics 2011*, 3(Suppl 1).

[Birkenheuer et al., 2012] Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., dos Santos Vieira, I., Fels, G., Gesing, S., Grunzke, R., Herres-Pawlis, S., Kohlbacher, O., Krüger, J., Lang, U., Packschies, L., Müller-Pfefferkorn, R., Schäfer, P., Steinke, T., Warzecha, K., and Wewior, M. (2012). Mosgrid: efficient data management and a standardized data exchange format for molecular simulations in a grid environment. *Journal of Cheminformatics*, 4(Suppl 1):21.

[Birkenheuer et al., 2011b] Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., Fels, G., Gesing, S., Grunzke, R., Herres-Pawlis, S., Kohlbacher, O., Krüger, J., Lang, U., Packschies, L., Müller-Pfefferkorn, R., Schäfer, P., Schuster, J., Steinke, T., Warzecha, K., and Wewior, M. (2011b). Mosgrid: Progress of workflow driven chemical simulations. In *Proc. of Grid Workflow Workshop 2011, Cologne, Germany*, volume 826.

[Gesing et al., 2011a] Gesing, S., Grunzke, R., Balasko, A., Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., Fels, G., Herres-Pawlis, S., Kacsuk, P., Kozlovszky, M., Krüger, J., Packschies, L., Schäfer, P., Schuller, B., Schuster, J., Steinke, T., Fabri, A. S., Wewior, M., Müller-Pfefferkorn, R., and Kohlbacher, O. (2011a). Granular security for a science gateway in structural bioinformatics. In *Proc. IWSG-Life 2011.*

[Gesing et al., 2012a] Gesing, S., Grunzke, R., Krüger, J., Birkenheuer, G., Wewior, M., Schäfer, P., Schuller, B., Schuster, J., Herres-Pawlis, S., Breuers, S., Balasko, A., Kozlovszky, M., Fabri, A. S., Packschies, L., Kacsuk, P., Blunk, D., Steinke, T., Brinkmann, A., Fels, G., Müller-Pfefferkorn, R., Jäkel, R., and Kohlbacher, O. (2012a). A single sign-on infrastructure for science gateways on a use case for structural bioinformatics. *Journal of Grid Computing*.

[Gesing et al., 2012b] Gesing, S., Herres-Pawlis, S., Birkenheuer, G., Brinkmann, A., Grunzke, R., Kacsuk, P., Kohlbacher, O., Kozlovszky, M., Krüger, J., Müller-Pfefferkorn, R., Schäfer, P., and Steinke, T. (2012b). The mosgrid community – from national to international scale. In *EGI Community Forum 2012*.

[Gesing et al., 2012c] Gesing, S., Herres-Pawlis, S., Birkenheuer, G., Brinkmann, A., Grunzke, R., Kacsuk, P., Kohlbacher, O., Kozlovszky, M., Krüger, J., Müller-Pfefferkorn, R., Schäfer, P., and Steinke, T. (2012c). A science gateway getting ready for serving the international molecular simulation community. *Proceedings of Science*, PoS(EGICF12-EMITC2)050.

[Gesing et al., 2011b] Gesing, S., Kacsuk, P., Kozlovszky, M., Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., Fels, G., Grunzke, R., Herres-Pawlis, S., Krüger, J., Packschies, L., Müller-Pfefferkorn, R., Schäfer, P., Steinke, T., Fabri, A. S., Warzecha, K., Wewior, M., and Kohlbacher, O. (2011b). A science gateway for molecular simulations. In *EGI (European Grid Infrastructure) User Forum 2011, Book of Abstracts*, pages 94 – 95.

[Grunzke et al., 2012] Grunzke, R., Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., Gesing, S., Herres-Pawlis, S., Kohlbacher, O., Krüger, J., Kruse, M., Müller-Pfefferkorn, R., Schäfer, P., Schuller, B., Steinke, T., and Zink, A. (2012). A data driven science gateway for computational workflows. In *Proceedings of the UNICORE Summit*.

[Grunzke et al., 2013] Grunzke, R., Breuers, S., Gesing, S., Herres-Pawlis, S., Kruse, M., Blunk, D., de la Garza, L., Packschies, L., Schäfer, P., Schärfe, C., Schlemmer, T., Steinke, T., Schuller, B., Müller-Pfefferkorn, R., Jäkel, R., Nagel, W., Atkinson, M., and Krüger, J. (2013). Standards-based metadata management for molecular simulations. *Concurrency and Computation: Practice and Experience*.

[Krüger et al., 2010] Krüger, J., Birkenheuer, G., Blunk, D., Breuers, S., Brinkmann, A., Fels, G., Gesing, S., Grunzke, R., Herres-Pawlis, S., Kohlbacher, O., Kruber, N., Lang, U., Packschies, L., Müller-Pfefferkorn, R., Schäfer, P., Schmalz, H.-G., Steinke, T., Warzecha, K.-D., and Wewior, M. (2010). Molecular simulation grid. In *6. German Conference on Chemoinformatics*.

[Krüger et al., 2014] Krüger, J., Grunzke, R., Gesing, S., Breuers, S., Brinkmann, A., de la Garza, L., Kohlbacher, O., Kruse, M., Nagel, W. E., Packschies, L., Müller-Pfefferkorn, R., Schäfer, P., Schärfe, C., Steinke, T., Schlemmer, T., Warzecha, K. D., Zink, A., and Herres-Pawlis, S. (2014). The mosgrid science gateway - a complete solution for molecular simulations. *Journal of Chemical Theory and Computation*, 10(6):2232 – 2245.

[Potamitis and Schäfer, 2014] Potamitis, I. and Schäfer, P. (2014). On classifying insects from their wing-beat: New results. *Ecology and acoustics: emergent properties from community to landscape, Paris, France.*

[Schäfer, 2014a] Schäfer, P. (2014a). Experiencing the Shotgun Distance for Time Series Analysis. *Transactions on Machine Learning and Data Mining*, 7(1):3 – 25.

[Schäfer, 2014b] Schäfer, P. (2014b). The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery.*

[Schäfer, 2014c] Schäfer, P. (2014c). Towards time series classification without human preprocessing. In *MLDM 2014*, volume 8556, pages 228 – 242.

[Schäfer, 2015] Schäfer, P. (2015). Scalable Time Series Classification. *DMKD.*

[Schäfer and Dreßler, 2013] Schäfer, P. and Dreßler, S. (2013). Shooting Audio Recordings of Insects with SFA. In *AmiBio Workshop, Bonn, Germany.*

[Schäfer and Högqvist, 2012] Schäfer, P. and Högqvist, M. (2012). SFA: A Symbolic Fourier Approximation and Index for Similarity Search in High Dimensional Datasets. In *EDBT*, pages 516 – 527.

Scientific Talks

17.06.2014 On classifying insects from their wing-beat: New results. Ecology and acoustics: emergent properties from community to landscape, Paris, France.

21.07.2014 Towards time series classification without human preprocessing. 10th International Conference, on Machine Learning and Data Mining in Pattern Recognition (MLDM) - St. Petersburg, Russia.

13.07.2013: Shooting Audio Recordings of Insects with SFA. International AmiBio Workshop, Bonn, Germany.

28.03.2012: SFA: a symbolic Fourier approximation and index for similarity search in high dimensional datasets. 15th International Conference on Extending Database Technology (EDBT), Berlin, Germany.